Word2vec and its application to examining the changes in word contexts over time

Taneli Saastamoinen

Master's thesis
10 November 2020
Social statistics
Faculty of Social Sciences
University of Helsinki

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

Tiedekunta – Fakultet – Faculty Faculty of Social Sciences		Koulutusohjelma – Utbildingsprogram – Degree Programme Social statistics
Tekijä – Författare – Author Taneli Saastamoinen		
Työn nimi – Arbetets titel – Title Word2vec and its application to examining the changes in word contexts over time		
Oppiaine/Opintosuunta – Läroämne/Studieinriktning – Subject/Study track Statistics		
Työn laji – Arbetets art – Level Master's thesis	Aika – Datum – Month and year November 2020	Sivumäärä – Sidoantal – Number of pages 88

Tiivistelmä – Referat – Abstract

Word2vec is a method for constructing so-called word embeddings, or word vectors, from natural text. Word embeddings are a compressed representation of word contexts, based on the original text. Such representations have many uses in natural language processing, as they contain a lot of contextual information for each word in a relatively compact and easily usable format. They can be used either for directly examining and comparing the contexts of words or as more informative representations of the original words themselves for various tasks.

In this thesis, I investigate the theoretical underpinnings of word2vec, how word2vec works in practice and how it can be used and its results evaluated, and how word2vec can be applied to examine changes in word contexts over time. I also list some other applications of word2vec and word embeddings and briefly touch on some related and newer algorithms that are used for similar tasks.

The word2vec algorithm, while mathematically fairly straightforward, involves several optimisations and engineering tricks that involve tradeoffs between theoretical accuracy and practical performance. These are described in detail and their impacts are considered. The end result is that word2vec is a very efficient algorithm whose results are nevertheless robust enough to be widely usable.

I describe the practicalities of training and evaluating word2vec models using the freely available, open source gensim library for the Python programming language. I train numerous models with different hyperparameter settings and perform various evaluations on the results to gauge the goodness of fit of the word2vec model. The source material for these models comes from two corpora of news articles in Finnish from STT (years 1992-2018) and Yle (years 2011-2018). The practicalities of processing Finnish-language text with word2vec are considered as well.

Finally, I use word2vec to investigate the changes of word contexts over time. This is done by considering word2vec models that were trained from the Yle and STT corpora one year at a time, so that the context of a given word can be compared between two different years. The main word I consider is "tekoäly" (Finnish for "artificial intelligence"); some related words are examined as well. The result is a comparison of the nearest neighbours of "tekoäly" and related words in various years across the two corpora. From this it can be seen that the context of these words has changed noticeably during the time considered. If the meaning of a word is taken to be inseparable from its context, we can conclude that the word "tekoäly" has meant something different in different years. Word2vec, as a quantitative method, provides a measurable way to gauge such semantic change over time. This change can also be visualised, as I have done.

Word2vec is a stochastic method and as such its convergence properties deserve attention. As I note, the convergence of word2vec is by now well established, both through theoretical examination and the very numerous successful practical applications. Although not usually done, I repeat my analysis in order to examine the stability and convergence of word2vec in this particular case, concluding that my results are robust.

Avainsanat – Nyckelord – Keywords
word2vec, word vectors, word embeddings, distributional semantics, natural language processing, semantic change, vector representations

Ohjaaja tai ohjaajat – Handledare – Supervisor or supervisors
Matti Nelimarkka

Säilytyspaikka – Förvaringställe – Where deposited
Helsingin yliopiston kirjasto, Helsingfors universitets bibliotek, Helsinki University Library

Muita tietoja – Övriga uppgifter – Additional information

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

Tiedekunta – Fakultet – Faculty Valtiotieteellinen tiedekunta	Koulutusohjelma – Utbildingsprogram – Degree Programme Yhteiskuntatilastotiede	
Tekijä – Författare – Author Taneli Saastamoinen		
Työn nimi – Arbetets titel – Title Word2vec and its application to examining the changes in word contexts over time		
Oppiaine/Opintosuunta – Läroämne/Studieinriktning – Subject/Study track Tilastotiede		
Työn laji – Arbetets art – Level Pro gradu -tutkielma	Aika – Datum – Month and year Marraskuu 2020	Sivumäärä – Sidoantal – Number of pages 88

Tiivistelmä – Referat – Abstract

Word2vec on menetelmä niin sanottujen sanavektoreiden, tai sanaupotusten, laskemiseen luonnollisen tekstin pohjalta. Sanavektorit ovat sanojen kontekstien tiivistettyjä esitysmuotoja, missä kontekstit ovat alkuperäisestä tekstistä. Tällaisille vektoriesityksille on luonnollisen kielen käsittelyssä monia käyttökohteita. Ne sisältävät runsaasti informaatiota kunkin sanan kontekstista suhteellisen tiiviissä ja helposti käytettävässä muodossa. Sanavektoreita voidaan käyttää joko suoraan sanojen kontekstien tutkimiseen tai alkuperäisten sanojen informatiivisempina esitysmuotoina erilaisissa sovelluksissa.

Tässä työssä tarkastelen word2vecin teoreettista pohjaa, word2vecin soveltamista käytännössä ja sen tulosten arviointia, sekä word2vecin käyttöä sanojen kontekstien ajan yli tapahtuvien muutosten tutkimiseen. Esittelen myös joitakin muita word2vecin ja sanavektoreiden käyttökohteita sekä muita samanlaisiin käyttötarkoituksiin kehitettyjä ja myöhempiä algoritmeja.

Word2vec-algoritmi on matemaattiselta pohjaltaan suhteellisen suoraviivainen, mutta algoritmi sisältää useita optimointeja jotka vaikuttavat yhtäältä sen teoreettiseen tarkkuuteen ja toisaalta käytännön suorituskykyyn. Tutkin näitä optimointeja ja niiden käytännön vaikutuksia tarkemmin. Lopputulos on, että word2vec on hyvin suorituskykyinen algoritmi, jonka tulokset ovat kuitenkin riittävän vakaita ollakseen laajasti käyttökelpoisia.

Kerron työssäni word2vec-mallien sovittamisesta ja arvioinnista käytännössä, käyttäen vapaasti saatavilla olevaa avoimen lähdekoodin gensim-kirjastoa Python-ohjelmointikielelle. Sovitan useita malleja eri hyperparametrien arvoilla ja arvioin tuloksia eri tavoin selvittääkseni mallien sopivuutta. Lähdemateriaalina näille malleille käytän kahta eri suomenkielisten uutisartikkeleiden tekstikorpusta, STT:ltä (vuosilta 1992-2018) ja Yleltä (vuosilta 2011-2018). Kerron myös käytännön ongelmista ja ratkaisuista suomenkielisen tekstin käsittelyssä word2vecillä.

Lopuksi, käytän word2veciä sanojen kontekstien muutosten tutkimiseen ajan yli. Tämä tehdään sovittamalla useita word2vec-malleja Ylen ja STT:n materiaaliin, vuosi kerrallaan, jolloin annetun sanan kontekstia voidaan tutkia ja vertailla eri vuosien välillä. Kohdesanani on "tekoäly"; tutkin myös muutamaa muuta siihen liittyvää sanaa. Tuloksena on vertailu sanan "tekoäly" ja muiden liittyvien sanojen lähimmistä naapureista eri vuosina kullekin korpukselle, ja siitä nähdään että kyseisten sanojen kontekstit ovat havaittavasti muuttuneet tutkittuina ajanjaksoina. Jos sanan merkitys oletetaan jollakin tasolla samaksi kuin sen konteksti, voidaan todeta että sana "tekoäly" on eri aikoina tarkoittanut eri asiaa. Word2vec on kvantitatiivinen metodi, mikä tarkoittaa että mainittu semanttinen muutos on sen avulla mitattavissa. Tämä muutos voidaan myös kuvata visuaalisesti, kuten olen tehnyt.

Word2vec on stokastinen menetelmä, joten sen suppenemisen yksityiskohdat ansaitsevat huomiota. Kuten totean työssäni, word2vecin suppenemisominaisuudet ovat jo varsin hyvin tunnetut, sekä teoreettisten tarkastelujen pohjalta että useisiin onnistuneisiin käytännön sovellutuksiin perustuen. Vaikka tämä ei yleensä ole tarpeen, toistan oman analyysini tutkiakseni word2vecin suppenemista ja stabiiliutta tässä nimenomaisessa tapauksessa. Johtopäätös on että tulokseni ovat vakaat.

Avainsanat – Nyckelord – Keywords
word2vec, word vectors, word embeddings, distributional semantics, natural language processing, semantic change, vector representations

Ohjaaja tai ohjaajat – Handledare – Supervisor or supervisors
Matti Nelimarkka

Säilytyspaikka – Förvaringställe – Where deposited
Helsingin yliopiston kirjasto, Helsingfors universitets bibliotek, Helsinki University Library

Muita tietoja – Övriga uppgifter – Additional information

Contents

Chapter 1

Introduction

This thesis is about word2vec, which is a method of computational text analysis, and specifically its application to examining changes in the contexts of words over time. Word2vec [58, 60] is an algorithm for computing so-called word vectors, also known as word embeddings. These embeddings are useful in many different ways, as we will see. One thing they provide is a straightforward distance metric between words in the input text corpus, which distance is based on word co-occurrence in the original context. This distance can then be used to examine changes in word context over time, which arguably can be used as a reasonable proxy for changes in meaning over time. Another concept related to context and distance is synonymity.

One famous aspect of word embeddings produced with word2vec is their capability to organise concepts found in the original source material, without any guidance other than the word co-occurrences, i.e. the word contexts, in the original text. An example of this is depicted in figure 1.1. The model has learned that the relationship between Kreikka (Greece) and Ateena (Athens) is the same, i.e. in the same vector direction, as that between Suomi (Finland) and Helsinki. This picture was produced by using the UMAP dimensionality reduction technique [56] on vectors from a 100-dimensional word2vec model trained on year 2015 of the Yle corpus of Finnish news articles. Full details can be found in chapter 3. The original inspiration for the figure is [60].

1.1 Computational text analysis

There has been a great deal of interest in computational analysis of text in recent years. According to Grimmer and Stewart [30], who focus on analysing political writings, the primary problem that computational analysis aims to solve is the large volume of source material. Automated methods can help by making very large-scale text analysis possible. Although, as Grimmer and Stewart note, such computational methods cannot at present replace competent human readers as the primary analysers of texts, they can nevertheless offer supporting data for proving and disproving human-crafted hypotheses regarding the source material.

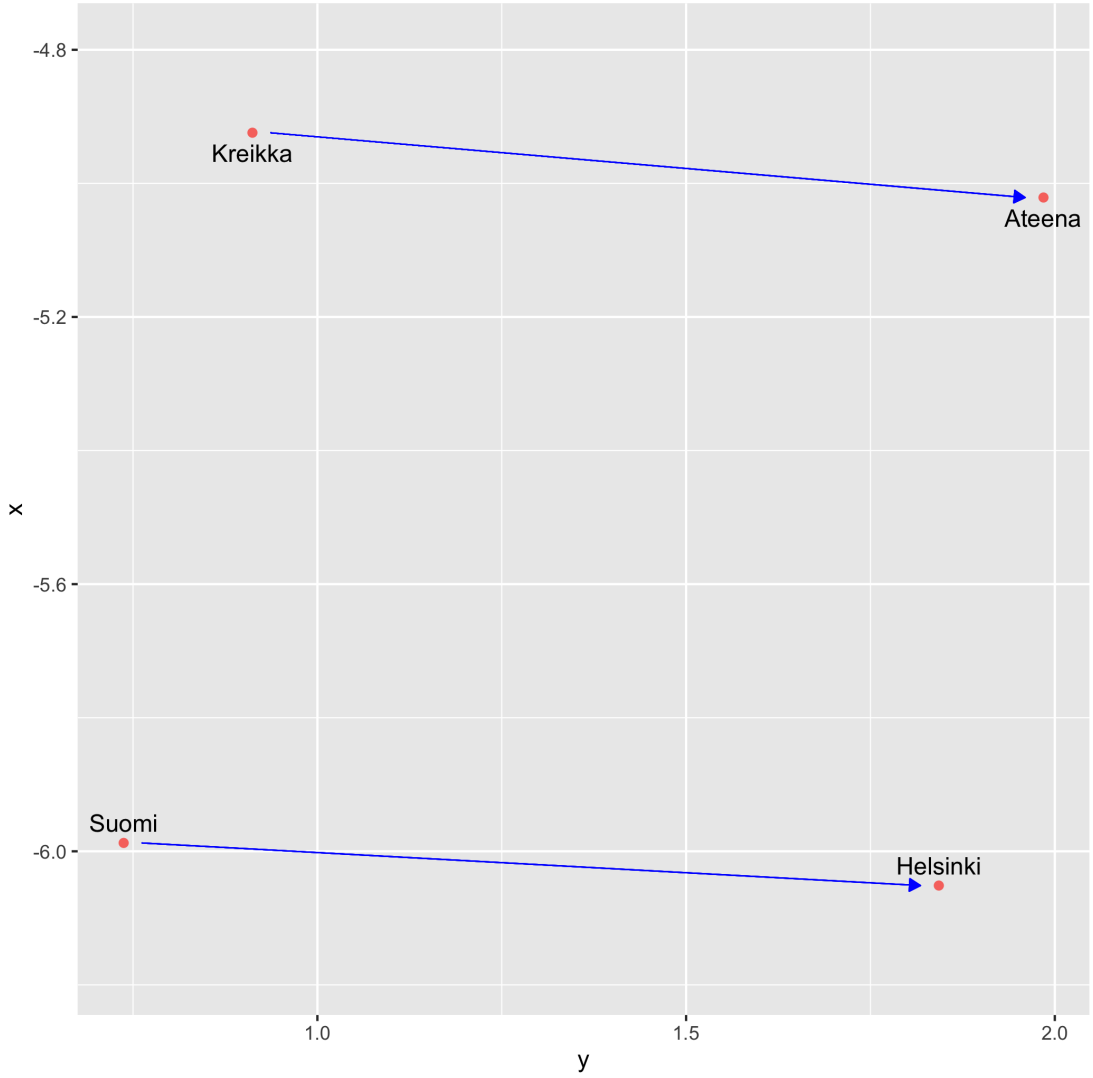

Figure 1.1: Two-dimensional UMAP projection of certain 100-dimensional word vectors created with word2vec. Source material: Yle corpus, year 2015. See text for details.

While computational text analysis can be useful, one must be aware of its limitations, and care must be taken to achieve good results with it. Grimmer and Stewart [30] discuss four useful principles for utilising automated methods. The first principle is to acknowledge that computational models are never perfect. This is due to both the current lack of profound understanding of exactly how humans generate text and the fact that any mechanical method must in any case fall well short of human-level understanding and intuition. This leads to the second principle, which is that mechanical models are no substitute for human understanding, but can be a useful supplement to it.

The third principle Grimmer and Stewart discuss is that no currently known method is the best one across the board in text analysis; rather, which model is the most useful depends on the task at hand. Finally, as the fourth principle, the results of any model used must be carefully validated in order to ascertain how valid the results are. Validation is not a trivial task, as it requires both technical knowledge of the model being used and relevant subject matter expertise of the "all-human" kind.

Another, similar perspective on these matters is provided by DiMaggio [16], who discusses a need to better understand the boundaries between human understanding and automated methods. He highlights the need to understand the strenghts and weaknesses of various analysis methods as they pertain to different types of corpus; the need to understand how to properly preprocess the data and the difference this can make to automated analysis; and the need to understand human bias and how this can affect the bias of automated techniques. Yet another discussion on the possibilities and dangers of computational text analysis is provided by Wilkerson and Casas [79]. They refer to various promising results obtained via automated methods, while warning of the difficulty in choosing a suitable model and validating the results.

An interesting case study in text analysis is Ylä-Anttila's dissertation [83], which brings together several papers of "classic" human analysis of political texts and one where automated methods, namely topic models, were used. Ylä-Anttila found the automated analysis useful and was able to use it as a basis for arguing for a possible interpretation of the data. In general, various researchers are looking into how to combine traditional qualitative methods with modern quantitative techniques of computational text analysis, as documented by e.g. Muller et al. [63]. This thesis, however, is mainly about the technical and practical details of word2vec, and such qualitative methods are outside our scope.

Turning to quantitative methods, we note that all current algorithmic natural language processing methods are computational and as such necessarily mechanical; they do not and cannot possess the kind of "intelligence" required to understand natural text in the same ways that humans do. This is why such automatic methods must always be accompanied by a qualitative human intelligence, and for this reason it is legitimate to ask how good such purely computational methods can possibly get at "understanding" human text. This topic has been widely studied since at least the 1980s [13, 18]. The following is a brief overview of some of the most well-known methods and the results achieved with them, with an eye towards how word2vec fits into this landscape.

1.2 Topic models

One of the first widely used natural language processing algorithms was latent semantic analysis, also known as latent semantic indexing (LSA, LSI) [13, 18]. Developed in the late 1980s, LSA belongs in the category of topic models, which

has since been expanded with other algorithms and techniques.

The goal of a topic model is to classify a collection of documents by estimating which topic, or topics, each of the documents is about. In the mathematical formulation, a "document" is simply an unordered collection of word frequencies, and a topic is a probability distribution over all the words in all the documents. For example, after fitting the model, one topic might have a high probability for the words "robot", "algorithm" and "program" (and a very low probability for other words), and another topic might have high probability for words like "robot", "industry" and "factory". A given document might then be estimated to be a mixture of 80 % of the "robot" topic and 20 % of the "industrial" topic.

In topic models, each document is represented as a so-called bag of words, which is simply a vector of word frequencies in the document. To produce this vector, we count the occurrences of each individual word in the document. Obviously with such a model the ordering and context of the original words is lost. The idea is that the word frequencies are very simple and quick to compute, and the resulting vectors are much smaller than the original documents, while the frequencies will hopefully still retain enough information about the semantic meaning of the original text.

In the original LSA topic model the word frequency vectors are computed in accordance with the bag of words technique and then "compressed" by using singular value decomposition on the matrix containing all document vectors. As is well known, with singular value decomposition, a good approximation of the original word-document matrix can be obtained by retaining only a relatively small number of the largest singular values and the vectors associated with them. For example, in the original LSA study the various data sets had a total vocabulary size of between 3000 and 6000 [18], while the authors were able to get decent results from LSA by retaining only the K=100 most meaningful vectors. In practice, the vocabulary size for natural text can often be in the tens of thousands or even hundreds of thousands, while for the value of K, something in the range of 100 to 500 is often sufficient. For instance, for another similar topic model called PLSA (probabilistic LSA), a value of K between 128 and 256 was found to be sufficient [37].

LSA's singular value decomposition is not the only way to tackle topic modelling. In the aforementioned PLSA [36, 37], the documents are handled quite differently – the bag-of-words frequency vectors are fitted with a latent-variable model. PLSA therefore models the topics directly using a probability distribution, whereas in LSA the estimated topic-to-word vectors are obtained as part of the singular value decomposition. The form of the result is the same in both cases, but PLSA provides topics which are easier to interpret, since it models topics as (non-negative) probability scores over words.

A newer topic model is latent Dirichlet allocation (LDA) from Blei, Ng and Jordan [4, 5]. LDA is an extension of PLSA into a full hierarchical Bayesian model, using the Dirichlet distribution as a prior for both the words and the topics, which are then modelled with multinomials as in PLSA. Although fitting this model

requires more complicated inference techniques, it gives better results than PLSA and suffers less from overfitting.

In general, the benefit of topic models is that they are computationally light and well understood, and their results, which are fairly straightforward to obtain, are often interesting. One caveat however is that the bag of words model is very crude and discards a lot of information, such as sentence-level context in its entirety. How much this matters depends on the application, but it is something to be aware of. The results of a topic model always require human interpretation and validation before they can be used. Ylä-Anttila ([83], article 4) provides a good description of a typical topic model workflow.

1.3 Word2vec and related methods

Topic models are based on the traditional concept of the vector space model, which was introduced earlier [17, 71]. The central idea, as was discussed, is to construct a representation of given text documents as dense vectors. Topic models are one way of doing this, and they proceed from the initial modeling choice of the bag-of-words model, in which each document is first condensed into a document vector.

Word2vec and other related algorithms use a more granular representation of the source text for their vector construction, namely, they operate on the sentence level: rather than representing an entire document as one vector, each sentence (regardless of which document it is from) is handled separately. Of course, one could seek even greater granularity than this, for example with character-based generative models, for which RNN-based techniques [74] are commonly used. Such very granular models are outside the scope of our discussion here.

Given a vector representation, at whatever granularity, the next task of a model is usually to seek a compressed representation of it. While LSA, for example, uses matrix factorisation on the word-document matrix to accomplish this, word2vec takes a different approach, being an autoencoder, i.e. a shallow neural network, as we will see. Such neural network approaches for learning word vectors have been investigated for some time [3]; word2vec is one of the major breakthroughs in this area. A more detailed historical review of topic models, word2vec and related models can be found in e.g. chapter 6 of the upcoming third edition of Jurafsky & Martin's book [42].

In what follows, we will first look at the theoretical underpinnings of word2vec in chapter 2. This chapter also includes discussion on some of its applications and on later, more advanced algorithms. How word2vec works in practice is examined in detail in chapters 3 and 4. Chapter 3 is dedicated to the practicalities of fitting a word2vec model and evaluating its performance, while chapter 4 describes an examination of two text corpora using word2vec, as an example of the kind of study that can be done with word2vec. Finally, chapters 5 and 6 summarise and discuss.

Chapter 2

The word2vec algorithm

Word2vec [58, 60] is a modern method for computing so-called word embeddings, also known as word vectors. Word embeddings are a way of encoding natural-language words into relatively low-dimensional, dense vectors, based on contextual information extracted from short sentences and sentence fragments. The idea is that these word embeddings contain information derived from the contexts of each target word, i.e. from the words frequently occurring near each target word, and are therefore more informative than the plain words by themselves. After computing the embeddings, they can then be used for various tasks such as clustering of words to detect semantic closeness and synonymy; they can also be used in more indirect ways, as we will see in section 2.5.

Word vectors and their efficient computation has been a topic of study since the 1980s [35]. One recent breakthrough was achieved by Bengio et al. [3], who proposed a neural network model for the computation. Word2vec is a continuation of this idea. Word vectors are an active topic of research; for example, another recent way of calculating word vectors is GloVe [66], which starts from a different viewpoint but arrives at a training objective similar to word2vec.

2.1 Skip-gram

There are two main variants of the word2vec algorithm, known as skip-gram and continuous bag of words (CBOW). In this section, I will first explain skip-gram in detail, and then explain how CBOW differs from it in the next section. The two variants are fairly similar in structure, and the hyperparameters of the algorithm (table 2.2; see below for details) are applicable to both variants.

To understand the computational objective of word2vec, it may help to consider the contexts of words in a more general setting. Given a corpus of natural text, one way to examine it is to consider the context in which each word appears, and one way to consider the context of each word is to construct a simple word-word co-occurrence matrix. An example is seen in table 2.1. Such a table records how many times each word in the vocabulary appeared in the same context, such as

	aardvark	banana	cream	...	eat	...	zombie
aardvark		0	0	...	12	...	0
banana	0		14	...	38	...	1
cream	0	14		...	19	...	2
⋮				⋱			
eat	12	38	19	...		...	13
⋮					⋱		
zombie	0	1	2	...	13	...	

Table 2.1: An example of a word-word co-occurrence matrix.

in the same sentence, as each other word. The context window can be taken to be the entire sentence, or as is usually done, an upper limit can be set for the window size within each sentence.

Co-occurrence matrices are common in natural language processing [41]. They can be viewed as a generalisation of the classic bag of words representation, which was used in pioneering natural language models such as LSA [13, 18]. The bag of words representation is essentially a word-document co-occurrence matrix. To obtain higher granularity, one can construct a word-word co-occurrence matrix, as shown above.

Word2vec, then, essentially computes a compressed representation of such a word-word co-occurrence matrix.

2.1.1 Input transformation and negative sampling

Word2vec is an autoencoder. This means it is a shallow neural network that tries to compute a structured, simplified encoding of the given input, in such a way that this encoding can be used to reconstruct the original signal (to a reasonable level of accuracy). In other words, an autoencoder attempts to find a good way of compressing its input and thus achieve dimensionality reduction.

The input of word2vec is a collection of sentences, or sentence fragments. These are simply ordered lists of words. Given a collection of sentences, the algorithm considers each word in its context, which is defined to be the previous C words and the following C words. The integer $C > 0$ is a hyperparameter; typical values are 5 or 10. This and other hyperparameters are described in table 2.2.

It must be noted that the output of word2vec is not meant to be an exact replica of the input. Rather, the output is a predicted frequency table of the context words for each input word, while the input is handled one example at a time; the output is therefore an aggregated view of the input seen so far.

In what follows "output" refers to the output of the last layer of the word2vec neural network. The result of the algorithm that we're actually interested in are the encodings, or word vectors, which after training are read off from the weight

Q	The total number of words in the input corpus. Only counts words that are considered given the threshold M (see below).
V	The vocabulary, i.e., the set of all words that we consider in the source material.
N	The size of the vocabulary, i.e. the number of unique words: $N = \|V\|$. Often hundreds of thousands for real-life text corpora.
D	The dimensionality of the desired encodings, i.e. the length of each resulting embedding vector. Usually good results are obtained with values in the 100–600 range.
C	The maximum size of the window, in each direction from the word in the middle. That is, the maximum sentence length used for training is $2C + 1$: the word in the middle and C words before and after. Usually this is set to 5–10.
M	The minimum number of occurrences required for a word to be included in the vocabulary. The purpose of this is to remove words which are too rare to be informative, before training the model. Common values are 3–10.
K	Negative sampling factor; see text for details. Common values are 5–10.

Table 2.2: The hyperparameters of word2vec.

matrix of the first hidden layer, rather than from the last layer. Therefore the "output" of the word2vec neural network is not the same as the "result" of the word2vec algorithm.

As is seen from this, the total number of parameters in a word2vec model can be quite large. For example, a realistic real-life data set might have some 92,000 unique words; if the desired embedding dimensionality is then set to, say, 300, the model will have $92,000 \times 300 = 27,600,000$ total parameters in the embedding layer (the network structure that results in this simple formula is detailed below). The embedding dimensionality is word2vec's most important hyperparameter, and it can have a drastic effect on bias on one hand and possible overfitting on the other hand. This is investigated in more detail in chapter 3.

The input to the word2vec algorithm is a set of sentences, which are simply ordered sequences of words or tokens. We assume that all necessary preprocessing is done by this point. Common preprocessing steps are lowercasing, removing excess punctuation and splitting the input into sentences and the sentences into words or tokens. More information about preprocessing in practice can be found in chapter 3.

There is a preliminary pass through the entire input corpus to determine the vocabulary. This is done by counting, for each word, how many distinct sentences it appears in, and then culling words which appear too rarely in the corpus to be of much interest. This culling threshold is a hyperparameter of the algorithm, which

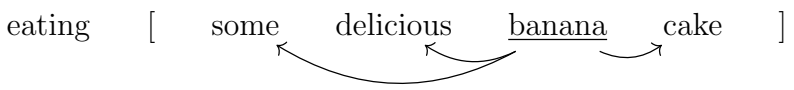

Figure 2.3: Example of word2vec skip-gram input transformation with $C = 2$, when processing the word "banana". Only the positive examples are depicted.

I call M (it is unnamed in the original paper); common values for M are 3–5. As is usual for such hyperparameters, there is no straightforward way to determine the "best" value of M; this depends on factors such as the total size of the corpus, and in a given situation one should experiment with various values of M to see which one gives the best results.

Once the vocabulary has been constructed, the word2vec model is trained on the corpus, which involves several epochs, i.e. several complete go-throughs of the entire corpus. The number of epochs is usually between 10 and 30. After a point, more epochs do not necessarily give better performance but may instead risk the model overfitting. This is elaborated on in chapter 3.

A full training epoch of the algorithm involves simply going through each input sentence and converting it into suitable input-output pairs, with which the neural network model is trained in the usual way, i.e. with gradient descent and backpropagation. The sentences can be processed in any order; indeed, a different random ordering for each epoch can be beneficial, as is often the case for neural network training (see e.g. [27], ch. 8). To speed up the training, the sentences, or the input-output pairs, may be collected into moderately-sized batches, for example of 500 sentences or 10,000 input-output pairs per batch. The optimal batch size depends on the implementation and the hardware used.

The method for converting of each input sentence into input-output pairs is shown in listing 2.1. In order to more easily understand this procedure, let us consider an example sentence. Let our context window size be $C = 2$ and say we are considering the sentence

$s =$ "eating some delicious banana cake".

If we are currently at the word "banana", then with a context window width $C = 2$ we would consider the context words "some", "delicious" and "cake", producing the input-output pairs of "banana" $\rightarrow$ "some", "banana" $\rightarrow$ "delicious" and "banana" $\rightarrow$ "cake", as depicted in image 2.3. Note that with this context window size we end up missing the connection between "banana" and "eating", which demonstrates the effect that the selection of context window width C can have.

The positive examples mentioned here are one half of the required material for training the word2vec model. The other half are the so-called negative examples.

The negative examples are produced with the method specified in listing 2.2. This part of the algorithm is, in the abstract, a softmax ([27], pp. 180–184). Softmax, a common technique in machine learning, is used as the final output step of

```
Input: Ordered list of words W (the sentence); list of words V (the
       vocabulary); list of word frequencies F; integer C > 0 (the context
       window size); integer K > 0 (negative sampling factor)
Output: List of triplets (x, y, Z), where x is the input word, y is the
        desired output word (i.e. positive example), and
        Z = (z_1, z_2, ..., z_K) are the negative examples
result ← ();
foreach w ∈ W do
    if w ∈ V then
        context ← (up to C words from W before w) + (up to C words
         from W after w);
        foreach c ∈ context do
            negatives ← NegativeSample (V, F, K, w, c);
            result ← result + (w, c, negatives);
        end
    end
end
return result
```

Algorithm 2.1: Word2vec input transformation, skip-gram version.

```
Input: List of words V (the vocabulary); list of word frequencies F;
       integer K > 0 (negative sampling factor); word w; word c
Output: List of negative examples Z = (z_1, z_2, ..., z_K)
result ← ();
for i ← 1 to K do
    repeat
        negative ← sample a word from V according to word frequencies F;
    until negative ∉ ({w, c} ∪ result);
    result ← result + negative;
end
return result
```

Algorithm 2.2: Word2vec negative sampling.

neural networks which perform multi-class classification, i.e. a classification task with more than two categories. Together with a corresponding loss function, the training objective is for the network to output the unique correct label for every input; all labels other than the one correct one are considered incorrect. The loss function most commonly used with softmax is the cross-entropy loss ([27], pp. 129–130).

With word2vec of course the task is not classification: instead of predicting which single word occurs in the context of the input word, our task is to provide a distribution of all the context words. If we tried to use regular softmax

and cross-entropy loss for this task, the positive and negative updates would work in opposite directions. In our example, with cross-entropy loss, the input-output pair of "banana" → "delicious" would cause the weights to be adjusted so that the output "delicious" became more likely, but the standard cross-entropy adjustment would make any other pair less likely, including the pair "banana" → "cake" which is an actually-occurring case in our example. Likewise, for the next pair, the adjustment that makes "banana" → "cake" more likely would in turn make "banana" → "delicious" less likely.

Perhaps more importantly, regular softmax is not suitable for situations with a very high number of possible "correct answers", such as softmax. As was mentioned, many real-life text corpora have a unique word count that reaches hundreds of thousands. With cross-entropy loss there would then have to be one "positive" adjustment of the weights and, say, 92,000 "negative" adjustments. While softmax works well for a small to moderate number of possible classes, say ten classes, or a thousand, it would be computationally very heavy with hundreds of thousands.

This brings us to negative sampling, which is the second of the two essential components of word2vec input transformation. With negative sampling, as seen in listing 2.2, we simply sample a small, constant number of negative examples, rather than using the entire vocabulary. This neatly solves the problems discussed above. It is not obvious a priori how well such a sampling approach will work, but in practice it turns out to work quite well.

In more detail, we first choose how many negative examples we want to sample for each positive example; this is the hyperparameter $K > 0$, an integer. Then, each time we convert a part of a sentence into an input-output pair, we also sample K negative examples from the entire vocabulary. This sampling is done from the unigram distribution, i.e. from the distribution $P(w_i) = U(w_i)/Q$, where word w_i's probability is set to be the total number of its occurrences in the corpus, $U(w_i)$, divided by the total number of words in the corpus Q. While sampling, we need to be careful to not include either the current input word or the current output word (i.e. positive example); we also need to avoid duplicates. A simple solution is to avoid such duplicates is to simply repeat the draw. This works well since, given that the vocabulary in real-world applications is usually massive and no word has a dominatingly large frequency in it, duplicates rarely occur and when they do, getting more than a few duplicates in a row is extremely unlikely.

2.1.2 Neural network architecture

In the previous section we saw how the input is transformed, in the skip-gram variant of word2vec, into raw material to be used for training the neural network. In this section we take a closer look at the neural network architecture. Both the input transformation and the neural network structure are different between the skip-gram and CBOW variants of the algorithm, but the basic building blocks are the same.

Let us consider first how word2vec would work as a "straight up" traditional

$$a^1 \coloneqq x_k W^{[1]} = [\overset{1}{0} \dots \overset{k}{1} \dots \overset{N}{0}] \begin{bmatrix} — & w_1^{[1]} & — \\ & \vdots & \\ — & w_k^{[1]} & — \\ & \vdots & \\ — & w_N^{[1]} & — \end{bmatrix} = w_k^{[1]}$$

(a) The one-hot-encoded representation of the input word, the kth in the vocabulary, is first multiplied by the first-layer weight matrix $W^{[1]}$, which simply selects the kth weight vector, $w_k^{[1]}$, from that matrix.

$$z^2 \coloneqq a^1 W^{[2]} = w_k^{[1]} W^{[2]} = [— \quad w_k^{[1]} \quad —] \begin{bmatrix} | & & | \\ w_1^{[2]} & \dots & w_N^{[2]} \\ | & & | \end{bmatrix} = [w_k^{[1]} \cdot w_1^{[2]} \dots w_k^{[1]} \cdot w_N^{[2]}]$$

(b) The vector $w_k^{[1]}$ is multiplied by the second-layer weight matrix $W^{[2]}$, which results in a vector z^2 of dot products between $w_k^{[1]}$ and each of the N column vectors in $W^{[2]}$.

$$a^2 \coloneqq \text{softmax}(z^2)$$

(c) The vector z^2 goes through softmax, resulting in the final output a^2.

Figure 2.4: Word2vec skip-gram architecture; the full version, i.e. without negative sampling.

neural network, without the optimisation of negative sampling; I will call this traditional architecture the full network. After describing this in detail it will then be easy to see how negative sampling works and how it provides a large performance boost.

An unoptimised neural network version of word2vec is given individual input-output pairs, and the network is trained based on these. This process is depicted in figure 2.4. Note that I am describing here a network architecture of type "xW" as opposed to "Wx", i.e. one where the input vector x is a row vector which is multiplied on the right by the weight matrix W, rather than the weight matrix being multiplied from the left by a column vector. These two possibilities are functionally equivalent, and which one is used generally depends on the lower-level implementation.

In step 1, fig. 2.4a, the input word is one-hot encoded in the usual way, and the resulting input vector of size $1 \times N$ is then multiplied by the first layer's weight matrix $W^{[1]}$ of size $N \times D$, producing a vector of size $1 \times D$. Because the input vector has the value 1 in only one position and is zero elsewhere, this operation

simply selects one of the row vectors in $W^{[1]}$, i.e. the D-dimensional embedding vector corresponding to the input word.

In step 2, fig 2.4b, this embedding vector is multiplied by the second layer's weight matrix $W^{[2]}$ of size $D \times N$, producing a vector z^2 of size $1 \times N$. In the same way as the first layer's weight matrix $W^{[1]}$ contains N embedding vectors each of length D, arranged into rows, the second layer's $W^{[2]}$ contains another N embedding vectors of length D each, arranged into columns. I will call these secondary embeddings, as opposed to the primary embeddings in the first layer. The main point is that, given an embedding vector a^1 which was selected from $W^{[1]}$, the vector-matrix multiplication $a^1 W^{[2]}$ computes the dot product $a^1 \cdot c_i$ for each column vector c_i in $W^{[2]}$, and stores these dot products in the resulting vector of length N. Let us call the result z^2.

In step 3, fig. 2.4c, we apply softmax ([27], pp. 180–184) to the vector z^2 to produce the output of the word2vec algorithm. The purpose of the softmax, as usual, is to convert the dot products into a predicted probability distribution; here the probability distribution represents our view of how likely any given word is to appear inside the same context window as the input word. Recall that in z^2 we have the dot product of the input word's embedding vector $v(w)$ and another embedding vector $v'(w_i)$, for all words w_i in the vocabulary; here $v(.)$ denotes the primary embedding vectors, i.e. the ones in the first layer, and $v'(.)$ refers to the secondary embedding vectors.

The softmax, then, concludes the forward step of the word2vec neural network. Next we do the backward step. For this, we need the "correct" answer for the original input word, which is provided by the input transformation. We calculate the value of the loss function, which depends on the difference between the model's current prediction and the expected answer, and use this to compute the values for the derivatives of the loss function, which are propagated backwards through the network in the usual way.

Let us elaborate on the mathematics at this point. The goal of the model is to estimate, or replicate, the probability distribution of the context words given an input word. We can represent this objective with the log likelihood function

$$\log \mathcal{L} = \sum_{(w,c) \in S} \log p(c|w), \tag{2.1}$$

where S is the set of all word-context word pairs, i.e. "input-output" pairs, that we obtain from the input transformation. As mentioned above, we model the probabilities $p(c|w)$ with a softmax involving the dot products between the respective embedding vectors; these embedding vectors are $v(w)$ for the input word w, and $v'(c)$ for each context word c, where $v(.)$ and $v'(.)$ refer to the primary and secondary embeddings respectively. We therefore arrive at the following representation:

$$p(c|w) = \frac{\exp(v(w) \cdot v'(c))}{\sum_{w' \in V} \exp(v(w) \cdot v'(w'))}, \tag{2.2}$$

where the sum in the denominator is over all words in the vocabulary. To be clear, in the forward step we first compute $v(w) \cdot v'(w')$ for every possible context word w' in the vocabulary and store these in a length N vector; we then use 2.2 to transform each of these dot products in that vector into a softmax-normalised probability distribution.

Given an ordinary softmax model such as this, the standard loss function to use is cross-entropy loss ([27], pp. 129–130). This takes the form

$$L(y, \hat{y}) = -\sum_{i=1}^{N} y_i \log \hat{y}_i \tag{2.3}$$

where y is the one-hot-encoded expected answer (a length N vector), $\hat{y}$ is the prediction of the model, i.e. the result of the softmax (also length N), and the sum is over all N positions in the two vectors. As is known, and as can be seen from this, the cross-entropy loss function expects the prediction to be "binary" in that a prediction of anything less than 1 in the "hot" position of the expected answer is penalised, as is a prediction of anything higher than 0 in the other positions. An important point here is that even though there are multiple "correct" answers – there are multiple context words that in fact occur – and thus we never want the model to end up giving its full weight to just one of them, the cross-entropy loss still works well in practice, since in machine learning the weight adjustments are done gradually with little "nudges", rather than all at once. The weights in the model are adjusted little by little, one example at a time and over several training epocs. Each time the weights are adjusted, rather than a full adjustment, only a small fraction of the computed adjustment is used: this learning rate is often in the range of 10^{-2}–10^{-5}, or in any case much smaller than one. More details about model training and learning rates can be found in standard machine learning textbooks such as Goodfellow et al. [27].

Putting equations 2.1 and 2.2 together, we arrive at a possibly clearer form of the training objective:

$$\log \mathcal{L} = \sum_{(w,c) \in S} \left[v(w) \cdot v'(c) - \log \sum_{w' \in V} \exp(v(w) \cdot v'(w')) \right]. \tag{2.4}$$

From either 2.2 or 2.4 we see that the computational cost of training this full model is directly proportional to N, the total size of the vocabulary, since the sum in the objective is taken over all context words in the vocabulary. This vocabulary size is often very large in practical applications – tens of thousands, even hundreds of thousands – and thus ends up dominating the computational complexity of the model. Mikolov et al. [58] note several previous attempts to make this part of the computation more efficient and introduce their own solution: negative sampling.

As we saw before, negative sampling involves using randomly drawn words as negative examples. This is in contrast to the ordinary softmax, which uses every word in the vocabulary, except the correct answer, as negative examples. Given the

vocabulary sizes in real world corpora, sampling a small, constant number of negative examples instead can obviously provide a large benefit; however, this method cannot be as accurate as calculating an update for each and every word, since under negative sampling we are simply ignoring a major fraction of the words. The question is, therefore, whether such a sampling strategy works and how large a sample is required. Mikolov et al. answered this in the affirmative and demonstrated good results with a negative sampling factor K between 2–20. Compared to a vocabulary of, say, a hundred thousand words, even the largest of these values, $K = 20$, means doing only 0.02 % of the work. This is a remarkable result which makes a massive difference to the efficiency of computing word embeddings with word2vec.

Above, we saw how negative sampling is implemented as a part of the input transformation. From the point of view of the neural network, negative sampling does not affect the architecture in any way, but simply provides a new way of computing the forward and backward steps in the second layer. Rather than computing the vector-matrix product of the embedding vector of the input word and the entire second layer weight matrix $W^{[2]}$, we compute only the dot products between the input embedding and the positive and negative examples. As with the full softmax, we have the positive example and a number of negative examples, and we want to adjust these in opposite directions; unlike the softmax, we do not compute a fully normalised probability distribution. Instead, we simply adjust each of the dot products, positive and negative, using a simple sigmoid function. Although this is somewhat questionable from a probability modelling point of view, the same logic applies that was previously described for the full softmax: we are in any case only taking small, even tiny, steps towards each "correct" answer, rather than utilising all of the computed "correct" answer completely. It turns out that in practice it suffices, for each input word, to move towards the correct answer, away from a handful of incorrect answers, and to ignore everything else.

To be more specific, under negative sampling we calculate the likelihood of each input word as a combination of simple unnormalised pseudo-probabilities. For the positive example, given an input word w and a positive example c, we compute the dot product $v(w) \cdot v'(c)$ and apply the standard sigmoid function $\sigma(z) = \frac{1}{1+\exp(-z)}$ to it to get the model's view of the probability that the context word c occurs in the context window of the input word w:

$$p^+(c|w) = \sigma(v(w) \cdot v'(c)) = \frac{1}{1 + e^{-v(w)\cdot v'(c)}}. \tag{2.5}$$

For the negative examples, we similarly calculate their probability as a combination of sigmoid-transformed dot products between $v(w)$ and each negative example z's secondary embedding $v'(z)$:

$$p^-(z|w) = \sigma(v(w) \cdot v'(z)) = \frac{1}{1 + e^{-v(w)\cdot v'(z)}}, \tag{2.6}$$

where z is a single negative example. To obtain the likelihood of an input word, we want to maximise p^+ while minimising p^-; this is equivalent to maximising

p^+ and maximising $1 - p^-$. This leads to the log likelihood function, for a single input-output pair (w, c) and its associated negative examples $Z = (z_1, z_2, \ldots, z_K)$,

$$\begin{aligned}
\log \mathcal{L}(c|w, Z) &= \log \left[p^+(c|w) \prod_{z \in Z} (1 - p^-(z|w)) \right] \\
&= \log p^+(c|w) + \sum_{z \in Z} \log(1 - p^-(z|w)) \\
&= \log \sigma(v(w) \cdot v'(c)) + \sum_{z \in Z} \log(1 - \sigma(v(w) \cdot v'(z))) \\
&= \log \sigma(v(w) \cdot v'(c)) + \sum_{z \in Z} \log \sigma(-v(w) \cdot v'(z)) \qquad (2.7) \\
&= \log \frac{1}{1 + e^{-v(w) \cdot v'(c)}} + \sum_{z \in Z} \log \frac{1}{1 + e^{v(w) \cdot v'(z)}}, \qquad (2.8)
\end{aligned}$$

where we made use of the fact that $1 - \sigma(q) = \sigma(-q)$. The forms 2.7 and 2.8 are equivalent but are both written down here for clarity.

Finally, in machine learning it is common to minimise a loss function rather than maximising a log likelihood function. The loss function can be defined very simply, as the negative of the log likelihood function:

$$\begin{aligned}
L(c|w, Z) &= -\log \mathcal{L}(c|w, Z) \\
&= -\log \sigma(v(w) \cdot v'(c)) - \sum_{z \in Z} \log \sigma(-v(w) \cdot v'(z)). \qquad (2.9)
\end{aligned}$$

Whether one minimises the loss or maximises the log likelihood, the end result is of course the same. As is seen, it is straightforward enough to calculate the derivative of the loss function for the purpose of backpropagation. Further details of this calculation are omitted. More information on the derivations specific to word2vec can be found in e.g. [25] and [69], and standard textbooks such as [27] are a good general reference for loss functions and their derivatives as used in machine learning.

2.2 Continuous bag-of-words

The other main variant of word2vec is continuous bag of words, or CBOW. As we saw, for skip-gram, the algorithm gets a single word as input and from it predicts the distribution of all other words in the context window. In CBOW, the algorithm gets as input all words in the context window other than the centre word, averaged together, and from this tries to predict the centre word.

2.2.1 Input transformation and negative sampling

A simple example of the CBOW input transformation is seen in figure 2.5. Note that all hyperparameters are the same as in the skip-gram variant, such as the

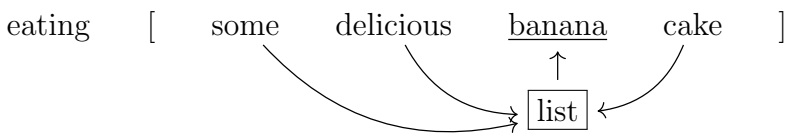

Figure 2.5: Example of word2vec CBOW input transformation with $C = 2$, when processing the word "banana". Only the positive example is depicted. The context words are collected into a list, which is the input, and the expected output is the centre word.

Input: Ordered list of words W (the sentence); list of words V (the vocabulary); list of word frequencies F; integer $C > 0$ (the context window size); integer $K > 0$ (negative sampling factor)
Output: List of triplets (X, y, Z), where $X = (x_1, x_2, \ldots, x_{2C})$ are the (up to) $2C$ input words, y is the desired output word (i.e. centre word of the window), and $Z = (z_1, z_2, \ldots, z_K)$ are the negative examples

```
1 result ← ();
2 foreach w ∈ W do
3     if w ∈ V then
4         context ← (up to C words from W before w) + (up to C words
           from W after w);
5         negatives ← NegativeSample (V, F, K, w, context);
6         result ← result + (context, w, negatives);
7     end
8 end
9 return result
```

Algorithm 2.3: Word2vec input transformation, CBOW version.

context window size C and the negative sampling factor K; for the full list see table 2.2.

The input transformation algorithm for CBOW is shown in listing 2.3. This is very similar to algorithm 2.1, the main difference being that it returns a list of context words as input and the centre word as output as described, which is the opposite of the skip-gram version. This difference is seen on line 6 of algorithm 2.3, as compared to line 7 of algorithm 2.1. Note that the negative sampling routine is here assumed to accept a list of words as its fifth argument, rather than a single word; other than this the negative sampling logic is unchanged from the skip-gram version.

$$
\begin{aligned}
x_C &\coloneqq \frac{1}{|O|}\sum_{k\in O} x_k = \frac{1}{|O|}\sum_{k\in O} [0 \dots \overset{k}{1} \dots 0] \\
a^1 &\coloneqq x_C W^{[1]} = [0 \dots \frac{1}{|C|} \dots 0 \dots \frac{1}{|C|} \dots 0] \begin{bmatrix} — & w_1^{[1]} & — \\ & \vdots & \\ — & w_k^{[1]} & — \\ & \vdots & \\ — & w_N^{[1]} & — \end{bmatrix} = \overline{w_C^{[1]}}
\end{aligned}
$$

(a) Let the set of one-hot-encoded context words be O; the size of this set is $1 \leq |O| \leq 2C$. For each context word in O, the corresponding weight vector is selected from the first-layer weight matrix $W^{[1]}$, and these weight vectors are averaged together into a^1. Note that it does not matter whether the averaging is done for the one-hot-encoded word vectors, as depicted here, or after the weight vectors are extracted from $W^{[1]}$. The result is a $1 \times N$ vector.

$$
z^2 \coloneqq a^1 W^{[2]} = \overline{w_C^{[1]}} W^{[2]} = [— \quad \overline{w_C^{[1]}} \quad —] \begin{bmatrix} | & & | \\ w_1^{[2]} & \dots & w_N^{[2]} \\ | & & | \end{bmatrix} = [\overline{w_C^{[1]}} \cdot w_1^{[2]} \dots \overline{w_C^{[1]}} \cdot w_N^{[2]}]
$$

(b) The vector $\overline{w_C^{[1]}}$ is multiplied by the second-layer weight matrix $W^{[2]}$, which results in a vector z^2 of dot products between it and each of the N column vectors in $W^{[2]}$.

$$
a^2 \coloneqq \text{softmax}(z^2)
$$

(c) The vector z^2 goes through softmax, resulting in the final output a^2.

Figure 2.6: Word2vec continuous bag-of-words (CBOW) architecture; the full version, i.e. without negative sampling.

2.2.2 Neural network architecture

The neural network architecture of the continuous bag of words edition of word2vec is depicted in figure 2.6. Note that this is the full, or unoptimised, version, i.e. one that uses a full softmax instead of negative sampling. In this version, the input is a list of context words, up to $2C$ of them, and the expected output is the centre word of the context window. In step 1 we first look up the embedding vectors corresponding to the input words; these embedding vectors are averaged to produce a single vector of length D, call it a^1. In step 2 we multiply a^1 by the second-layer weight matrix, i.e. the $D \times N$ matrix, producing a $1 \times N$ vector; and in the final step, we apply softmax to this vector, to produce the model's estimate

of the probability distribution of the expected output, i.e. of the centre word of the original window. As is seen, this process shares many similarities with that of the skip-gram version, the main differences being the slightly different natures of the input and output.

To be more precise about the input and output, in the CBOW case the log likelihood function of the model is

$$\log \mathcal{L} = \sum_{(w,C)\in S} \log p(w|C), \tag{2.10}$$

where S is now the set of all pairs of centre word w and corresponding context words $C = (c_1, c_2, \ldots, c_{2C})$. Similarly, the model computes the probabilities $p(w|C)$ using a straightforward softmax representation

$$p(w|C) = \frac{\exp(\overline{v(C)} \cdot v'(w))}{\sum_{w'\in V} \exp(\overline{v(C)} \cdot v'(w'))}, \tag{2.11}$$

where $\overline{v(C)}$ is the average of the embedding vectors $v(C)$ of the context word list C. These equations are of course very similar to 2.1 and 2.2, with just the inputs and outputs changed.

As with skip-gram, the full softmax is resource intensive to calculate also in the CBOW case: here too the sum in the denominator is taken across the entire vocabulary. Again, negative sampling can be used to reduce the computational load. Reasoning similarly as in the skip-gram case, we arrive at the likelihood function, for a single input-output pair (C, w) and its negative examples $Z = (z_1, z_2, \ldots, z_K)$,

$$\log \mathcal{L}(w|C, Z) = \log \sigma(\overline{v(C)} \cdot v'(w)) + \sum_{z\in Z} \log \sigma(-\overline{v(C)} \cdot v'(z)), \tag{2.12}$$

which is analogous to 2.7.

The primary benefit of the CBOW variant of word2vec over the skip-gram variant is that CBOW models are faster to train. This is because skip-gram transforms the input sentences into a relatively large number of input-output word pairs, whereas CBOW generates only one training example for each word in a given sentence. To be more precise, given a sentence of length L and a maximum context window size C, skip-gram transforms each of the L words into up to $2C$ input-output pairs, whereas CBOW transforms each of the L words into one input-output pair. Although the CBOW inputs are lists of up to $2C$ context words, and although the forward pass and backpropagation therefore do a similar amount of computation, the backpropagation uses the same correction for all context words in the CBOW case, rather than calculating separate corrections for each as with skip-gram. In practice this difference turns out to be noticeable, resulting in smaller training times for the CBOW model ([58], pp. 8–9).

A drawback of CBOW is that the faster computation time is gained at the expense of accuracy, precisely since the backpropagation updates are averaged over

a number of words. It is not obvious exactly how large an effect this has. Mikolov comments informally [57] that skip-gram may have an advantage with smaller amounts of training data, since it can more effectively compute the updates for each individual word. CBOW on the other hand may have slightly better accuracy for the frequent words in the input corpus while also being faster to train. Mikolov cautions though that such judgements must be made on a case by case basis after experimenting with each option to see how well they end up working in practice, which is solid advice in general.

2.3 Optimisations and adjustments

While negative sampling is by far the most important optimisation technique introduced in the original word2vec papers [58, 60], there are several other tweaks and optimisations that the authors found make a noticeable difference.

First, in the skip-gram case, the hyperparameter C denotes the maximum size of the context window on each side. One adjustment to the algorithm is to, for each input sentence, draw the actual size R of the context window from the uniform distribution between 1 and C, i.e. $R \sim \text{Unif}(1, C)$. As this is done for each input sentence in turn, the algorithm ends up choosing words which are far away from the centre word less often, whereas words right next to the centre word are always included, since $R \geq 1$. The authors do not report whether and how much this procedure helps with the accuracy of the model, but it does provide a boost in performance, since the context window sizes end up, on average, smaller.

Another somewhat heuristical optimisation is the subsampling of frequent words. While the removal of stop words is a commonly used preprocessing technique for natural text [22, 55], the word2vec authors prefer subsampling, which accomplishes a similar function. The idea is that, when processing an input sentence, each word in the context window is discarded with a probability proportional to its total frequency in the corpus. That is, when processing an input sentence, a context word w_i is discarded with probability

$$P_d(w_i) = \max\left(0,\, 1 - \sqrt{\frac{t}{f(w_i)}}\right), \tag{2.13}$$

where t is a constant and $f(w_i)$ is the total frequency of the word w_i in the entire input corpus. It has to be noted that 2.13 is merely a heuristic aid, which the original authors have empirically found to be helpful, rather than a full probability distribution.

For example, if t is chosen to be 10^{-5}, 2.13 will result in a rejection probability of zero for words whose frequency in the corpus is also 10^{-5}. For words less frequent than this, the rejection probability remains zero, while for words more frequent than the threshold t, the probability that the word is rejected climbs fairly rapidly towards 1. By choosing t in accordance with the characteristics of the input corpus, words that are "too common" in the input, which are presumably stop words, can

thereby be discriminated against so that not too much processing power is spent on them. It should be noted that setting t too high, i.e. so high that very few words in the corpus have a frequency close to t, should not massively affect the quality of the results, but merely make it so that the training takes longer, since subsampling will then occur relatively rarely. However, setting t very low compared to the corpus frequencies will cause many words to be rejected in the subsampling, which can affect the quality of the results.

An important target for optimisation is the distribution used for negative sampling. The most natural distribution to use is the unigram distribution, in which each word's sampling probability is set to its relative frequency, i.e. $P(w_i) = U(w_i)/Q$ for a given word w_i, where Q is the total number of (in-vocabulary) words in the corpus and $U(w_i)$ is the total number of occurrences of w_i in the corpus. The word2vec authors experimented with various other distributions and concluded that the best-performing one was a variant of the unigram distribution where the unigram frequency is raised to the 3/4rd power, i.e.

$$P(w_i) = C\frac{1}{Q}U(w_i)^{3/4}, \tag{2.14}$$

where C is a normalising constant. It is of course straightforward to change the sampling distribution used in negative sampling, so this optimisation is easy to make.

Finally, negative sampling is not the only thing that can be used to make word2vec more performant. The authors also describe the alternative of hierarchical softmax [60]. Hierarchical softmax is a better-performing version of softmax that utilises a binary tree, so that only $O(\log_2(N))$ nodes need to be evaluated in order to obtain the full context distribution, rather than all N nodes as with ordinary softmax. In practice, it appears that although hierarchical softamax performs well and is fast to train, it is not very commonly used with word2vec: negative sampling is much simpler to implement and also gives excellent results.

2.4 Limitations

Having discussed how word2vec works in detail, it is instructive to consider also what it cannot do. As with any other model, word2vec has some inherent limitations, which are due to the assumptions made in the specification of the model.

The word2vev model is trained using a somewhat simplified view of the input sentences. As we saw before, in both main variants, skip-gram and continuous bag-of-words (CBOW), word2vec ignores the ordering of the words in the context window. This is fine for the main purpose of word2vec, which is concerned with constructing a useful representation of the context distribution; for this distribution the word order does not matter. However, this makes word2vec unsuitable for tasks such as natural language generation, for which more complex models such as RNN-based techniques [74] or the more recent BERT (Bidirectional Encoder Representations from Transformers) [14] are a better fit.

Word2vec also takes the words, or tokens, very "literally", in the sense that it builds one representation for each unique token in the source material. This works fine for many use cases, but it means word2vec cannot handle polysemy (words that have more than one meaning) very well, since all meanings of a token have the same embedding vector [2]. To get around this, one approach is to utilise part-of-speech tagging, which can be applied to the source material first, after which the word2vec embeddings are learned from the now-tagged material [76].

Because of the one-to-one mapping between literal tokens and embedding vectors, word2vec has difficulties with languages such as Finnish, where it would be desirable to treat various inflected forms of words as the same word even though the literal forms differ. To deal with this, it can be helpful to first lemmatise the source material, as we will see in chapter 3.

One interesting consideration is how well word2vec works with words that are rare in the source material. This is explored more in chapter 4, where a real-world example is considered and evaluated. As was said before, when the skip-gram variant of the algorithm is used, the algorithm can take full advantage of each individual occurrence of any given word. We will see that in this case word2vec can perform quite well even with low word frequencies.

One important caveat related to the practical usage of word embeddings is that, since word embeddings are derived from natural text, they will also reflect any biases present in the original text, such as gender bias and ethnic bias. This was noted by Bolukbasi et al. [7] and has been further studied by various authors such as Caliskan et al. [9]; for example, [23] investigates the bias-related differences in word embeddings over time. Methods have since been developed to reduce or eliminate such bias, but this has seen mixed success [26]. There is current research to try to better define and measure such bias in word embeddings, and thereby to reduce it [65]; these efforts are ongoing.

From a more technical point of view, the convergence properties of word2vec deserve some consideration. Word2vec, as a stochastic technique, is meant to converge eventually to the true distribution, or at any rate get closer to it as training progresses. The true distribution here is the context distribution, which is what word2vec approximates. However, the convergence properties of word2vec are not obvious.

In general, the key questions with regard to convergence are bias (or lack thereof) and speed. These have been examined by various authors since word2vec was published. Word2vec's central optimisation is negative sampling, which is essentially a simplified version of noise contrastive estimation (NCE) [31]. While the full NCE objective is well understood, Dyer [19] notes that word2vec's negative sampling by itself does not have the same strong consistency guarantees as NCE's objective. Other authors have pointed out that word2vec's negative sampling objective can be understood as equivalent to a matrix factorisation technique [51, 52], and of these Li et al. [52] arrive at the conclusion that word2vec with negative sampling, when viewed as matrix factorisation, is indeed guaranteed to converge.

As for the speed of convergence, direct analysis is again difficult, but empirical

evidence points to fairly rapid convergence, i.e. fast enough that it is not a problem in practice.

2.5 Applications of word2vec

Since word2vec, as a model, does not make predictions, it is not as directly applicable for practical tasks as models that do. Often word2vec is instead used as an intermediate step in a more involved machine learning pipeline. For example, in a task of natural language processing, one can use the word embeddings as input instead of the words themselves. Since word2vec's word embeddings contain information not only about each word, but also about its most significant neighbours, the data that is fed to the actual prediction task is therefore richer than the raw words would be. The prediction task in question can then benefit from this additional information: if two words are in some sense close to one another, their corresponding embeddings will also be similar (and vice versa), and this information can be utilised by the actual prediction task. This section gives some examples of such applications for word2vec embeddings.

One early example of a useful application of word2vec, by the original authors, is using the obtained vector representations to help with machine translation [59]. This is a good illustration of how helpful it is to have a method for building rich, informative vectors for each word in the input corpus: it enables one to not only compare words to other words, but also to compare words across languages. In chapter 4 we will examine a similar application, namely of comparing words to earlier versions of themselves, i.e. comparing word vectors based on a given year of material to the vectors obtained from earlier material.

Another intuitive use for word2vec's word embeddings is to use them to examine changes to word contexts over time. By training separate word2vec models for, say, each year of source material, one can gauge how the usage of a given word has changed over time. Such studies have been done by many authors [23, 32, 45, 47, 48, 81]. We will have much more to say on this topic in chapter 4, where our goal is to replicate such an experiment with different source material.

As we saw, word2vec embeddings are constructed from word co-occurrence data in a given corpus, and as such, the embeddings lend themselves naturally to investigations related to such co-occurrences. For example, Leeuwenberg et al. [50] develop a method of extracting synonyms from text using word2vec embeddings in a minimally supervised way. For this they use a new metric, relative cosine similarity, which, given a target word embedding w, compares the cosine similarity of the embeddings of each of w's nearest neighbours w' to those of the other nearest neighbours. The authors are also able to improve their system by adding part of speech tagging, which is an example of how word2vec embeddings can be used alongside other techniques as part of a larger system.

In another study of co-occurring words, Cordeiro et al. [11] have investigated whether word embeddings produced by word2vec, GloVe and other models are us-

able for distinguishing between compound phrases which are literal and those that are proverbial, or idiomatic, answering the question in the affirmative. Examples of such phrases are "credit card", clearly quite literal, and "ivory tower", not literal at all. The authors found a high correlation between automatically generated judgements of literalness and judgements provided by human experts.

Mitra et al. [61, 64] present some interesting findings on how to utilise word2vec for improving search engine results. As we described before, the word2vec architecture consists of two sets of embeddings, from which only the first embeddings are used as the result of the algorithm. However, the authors found it valuable to consider also the second set of embeddings, which they call "output" embeddings, as opposed to the first-layer "input" embeddings. While the primary embeddings are useful for a wide variety of tasks, it turns out that the secondary embeddings also encode useful contextual information, which can then be used for, for example, information retrieval and search engine queries.

Some impressive recent results have been obtained by applying word2vec to scientific literature. Tshitoyan et al. [77] report on using word2vec for knowledge discovery in the field of materials science. In their case, a word2vec model trained on 3.3 million scientific abstracts related to materials research was able to find latent, implicit knowledge in the input, and the model could then be used to easily summarise and recover this latent knowledge. Another, earlier study is due to Gligorijevic et al. [24], who use a custom model based on word2vec to automatically analyse 35 million patient records in order to discover co-occurring diseases and disease-gene relations. There are many other similar efforts, e.g. [84] to mention just one. A survey of such novel machine learning and deep learning approaches to biology and medicine, including word2vec and related models, can be found in [10].

2.6 After word2vec

As noted previously in section 2.4, one significant drawback of word2vec is that it assigns precisely one embedding vector to each token in the input, which makes it difficult to handle languages with many inflected forms, e.g. agglutinative languages such as Turkish or Finnish. One idea for improvement then is to model each token, or word, with more than one embedding vector, in order to increase the information available for the model. This is what the authors of FastText have done [6, 40]. With the FastText method, the model learns representations of character n-grams, i.e. subwords, and represents the tokens of the source material as the sum of several such n-gram vectors. The authors report that this method gives better results than traditional word2vec on various tasks, such as word analogy tasks.

Another direction of extending word2vec is to get the model to learn additional vectors related to each paragraph, i.e. to larger portions of text. This approach, called paragraph vectors [49], aims at building a more context-aware set of vectors

for the given input text. These vectors can then be used, for each paragraph, to predict the next word, given the word vectors of previous words as well as the relevant paragraph vector. The authors report some success in using this model for tasks such as sentiment analysis. Notably, their results are better than those obtained with more traditional methods such as bag-of-words models and support vector machines.

A rival method of constructing word vectors is GloVe, which stands for Global Vectors [66]. GloVe starts from a viewpoint similar to that of word2vec, i.e. from the word co-occurrence matrix, but performs a different analysis. Importantly, GloVe takes the global co-occurrence counts of words into account explicitly and proceeds from there, whereas word2vec trains its model one example at a time in a more stochastic manner. The vectors produced by GloVe are used in the same way as word2vec embeddings. It is difficult to predict which ones would perform better in any given task, but the results seem to be more or less comparable, whereas the word2vec model has the advantage of being easier to train.

In recent years there has been much interest in various algorithms and models that utilise embeddings, no doubt bolstered by the success of word2vec. For example, Wu et al. have introduced an open-source project called Starspace [80], which is a generalised system for training many different models involving embeddings. While StarSpace can be used to train word2vec models, it also has much wider applicability. In a similar vein, Grbovic and Cheng [29] report of a modified word2vec algorithm that they use for creating embeddings based, not on documents and words, but rather user sessions and user clicks on the Airbnb website. These embeddings are then used for content ranking and personalisation. As well, Wang et al. report of another custom algorithm based on word2vec that is used to power a recommender system on the Chinese e-commerce website Taobao [78]. Again, there are many other examples of such novel systems, too numerous to list here.

Finally, the current state of the art in many natural language processing (NLP) tasks is BERT, which stands for Bidirectional Encoder Representations from Transformers [14]. BERT is a complex model in which similarities to earlier works can be seen, as it incorporates sub-word embeddings, somewhat similar to FastText, as well as segment and position embeddings, somewhat similar to paragraph vectors, among other innovations. The result is a very versatile model for building accurate embeddings, which can then be used in many NLP tasks, such as question answering, sentiment analysis, semantic similarity, next sentence prediction and others.

BERT, being a very large deep learning model with dozens of layers (the original paper describes two model architectures with 12 and 24 layers respectively), does not bear much resemblance to the earlier, simpler word2vec model. Both however are neural networks used to construct word embeddings based on word contexts, and in this sense an understanding of word2vec can perhaps be helpful for an intuition of the current bleeding edge of research as well.

Chapter 3

Word2vec in practice

There are various ways to utilise the word vectors generated by word2vec. One application is to examine the evolution and change of word semantics over time, as was done by Kim et al. [45] and by Hamilton et al. [32] using Google Books data from between 1850–1999 and 1800–1999 respectively. Of course, similar studies could be done with other corpora and other languages. As an experiment, I sought to replicate what Hamilton et al. [32] had done, but using a large collection of news articles in Finnish as my source material; that is, I trained various word2vec models on Finnish-language corpora and examined the resulting word vectors to see what insights could be gained regarding word semantics. I used the open source library gensim [68] to train the word2vec model.

3.1 Source material and processing

As source material I used two corpora of Finnish-language news articles published on the internet: articles published by Yle, the Finnish public broadcaster, between 2011–2018 [82]; and articles published by STT, a Finnish news agency, between 1992–2018 [73]. These corpora are available from the Finnish Language Bank for non-commercial research use.

The relevant sizes of the corpora used are depicted in figure 3.1 and table 3.2. Notice that there are various ways of calculating the word count of a corpus in the context of word2vec. The count used here only takes into account the effective words in each year's subcorpus, i.e. only those words are counted that appear in at least $M = 3$ distinct sentences. This is because words more rare than the selected threshold M are not used in training. It is seen that, except for the relatively small size of the STT corpus between 1992 and 1997, the data sizes across years and across corpora are very similar, with an average of some 17–18 million words per year of material.

Number of words by year (1992–2018), STT corpus					
Minimum	First quartile	Median	Average	Third quartile	Maximum
9,426,788	13,920,816	17,020,988	17,327,603	20,869,238	23,811,267

Number of words by year (2011–2018), Yle corpus					
Minimum	First quartile	Median	Average	Third quartile	Maximum
15,354,338	16,892,120	18,822,041	18,705,345	20,113,790	22,038,975

Table 3.2: Word counts of corpora used.

For each corpus, I trained a separate word2vec model for each year, so that I could compare the results for certain words of interest across the years. This chapter gives a detailed description of the preprocessing and modelling as well as the results.

3.1.1 Data processing pipeline

I preprocessed the material by removing hyperlinks and dividing the text into sentences, using mainly full stops, and either removing all other punctuation or converting it into full stops. After this each sentence was lemmatised using the open-source LAS lemmatiser [62]. Finally, the resulting tokens were converted to lowercase. An example of preprocessing is given in tables 3.3, which shows the original formatting of the text, and 3.4, which shows the result of the preprocessing.

It must be noted that lemmatisation is not ordinarily used as a preprocessing step with word2vec, at least not for English. However, the situation is different when the source material is in Finnish, due to the complexity of the Finnish language. The issues stemming from this complexity are described in detail in the following section.

One common preprocessing technique is the removal of so-called stop words [22, 55]. In the original word2vec papers [60], as in gensim's word2vec implementation [68], a straightforward subsampling of frequent words is used instead, as detailed in chapter 2. The advantage of subsampling is that it is an automated method which is based on the calculated frequencies of words in the corpus, i.e. a separate list of stop words need not be provided in advance. I did not therefore remove the stop words in preprocessing, relying instead of the subsampling to take care of it.

After preprocessing, lemmatising and lowercasing the material, it was ready to be used for training word2vec models.

```
{
  "text": "Sisäministeriön kansliapäällikkö Päivi Nerg [arvioi
sunnuntaina](http://yle.fi/uutiset/8457147), että
sisärajakontrollit saattavat palata Schengen-alueelle.",
  "type": "text"
},
```

Table 3.3: An excerpt of the original Yle article data, in JSON [8] format.

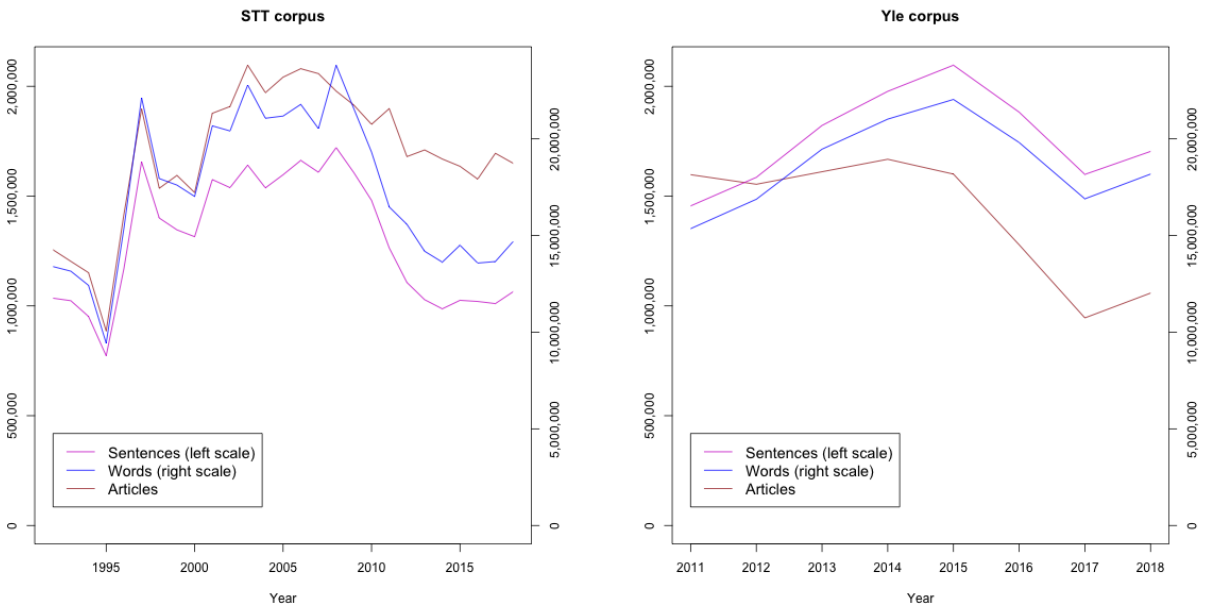

Figure 3.1: Sizes of corpora used.

```
sisäministeriö kansliapäällikkö päivi nerg arvioida
sunnuntai että sisä-rajakontrolli saattaa palata
schengenalue
```

Table 3.4: The preprocessed and lemmatised sentence corresponding to the input data from table 3.3.

3.1.2 Processing of Finnish-language text

While substantial progress has been made in recent years (see e.g. [34]), the Finnish language remains difficult to process algorithmically due to its complexity. Luckily word2vec as a technique does not require precise computational understanding of the semantics of language, such as part-of-speech tagging. Indeed, even relatively straightforward and common preprocessing techniques such as stemming or lemmatisation are not necessarily recommended for word2vec. Word2vec is capable of detecting the semantic relationships between token pairs such as “quick” : “quickly” :: “slow” : “slowly” [58, 60], whereas stemming or lemmatisation would transform a token like “quickly” into its base form “quick”, making this impossible.

However, the situation is somewhat more complicated when applying word2vec to Finnish. Finnish is an agglutinative language [39]. In practice this means that most Finnish words are routinely inflected, compounded and so on, which results in a great variety of different character strings or tokens that all nevertheless have the meaning of their base form. The sheer variety of these different “versions” of words makes automatic processing of Finnish more difficult than of, say, English,

Finnish word	Occurrences	Approximate English translation
suosittelee	8991	**recommends**
suositellaan	3456	is/are **recommended**
suositella	2892	to **recommend** (base form)
suositteli	2167	he/she/it recommended
suosittele	1582	does not/do not recommend
suosittelevat	1306	they recommend
suositellut	1232	he/she/it has recommended
suosittelen	1048	I recommend
suositeltavaa	854	it is recommended that
suosittelemaan	523	to recommend (e.g. in order to)
suositeltu	509	recommended
suositeltiin	460	was recommended
suositteleekin	411	he/she/it indeed recommends
suosittelivat	375	they recommended
suosittelemme	367	we recommend
suositelleet	345	they had recommended
suosittelisi	326	he/she/it would recommend
suositeltava	314	[a thing] to be recommended
suosittelisin	190	I would recommend
suosittelema	186	a recommended [thing]

Table 3.5: An example of inflected Finnish word forms and their approximate English translations. Corresponding unique English word forms are bolded.

since the number of superficially different tokens which yet have the same meaning is so large. The common solution to this problem is to preprocess the tokens in the source material by either stemming or lemmatising them. For Finnish, lemmatisation has been shown to give better results [46]. While lemmatisation substantially alters the tokens that are used as input for various language processing tasks, and thus potentially changes the results, it has been shown that in a statistical sense at least the underlying word frequency distributions are not majorly altered by lemmatisation, even for Finnish [12].

The following simple example illustrates the need for lemmatisation for Finnish, which arises from word inflection. I selected arbitrarily the word "suositella" ("to recommend") and counted the total number of its occurrences across both corpora (STT and Yle). Table 3.5 lists the 20 most common inflected forms of this word in the material, along with a free-form English translation of each. The main word, "recommend", occurs in only three different inflections in English, whereas the corresponding Finnish word "suositella" has hundreds of different inflected forms – too numerous to list them all.

As was mentioned, lemmatisation provides a solution to this problem. For example, the open-source LAS lemmatiser [62], which I used, is able to lemmatise

each of the inflected forms in table 3.5 into the base form. It has to be noted though that LAS's lemmatisation capability is not perfect; for example, the inflected compound word "tehdasvalmisteisille" (lit. factory-manufactured) is lemmatised into the nonsensical "tehdasvalmisteisä" (lit. factory-manufacture-father). The occasional inability to handle such edge cases is a well-known limitation of contemporary language tools for Finnish.

Another related issue in processing Finnish text are compound words. As is known, compound words are very common in Finnish [38]; they basically consist of two (or more) words spelled together without spaces – often two nouns, but other combinations such as adjective-noun occur with reasonable frequency. Given that compound words are very common, and that it is possible and occasionally desirable to create new compounds ad hoc e.g. to create emphasis, these compound words may then end up frivolously enlarging the vocabulary, similar to the inflected words discussed above.

To give a concrete example, consider the compound word "tietokonealgoritmi" ("computer algorithm"), occurring in the Yle material from 2011. This is literally "tietokone+algoritmi", i.e. "computer+algorithm", spelled as one word; however, the word "tietokone" ("computer") is itself also a compound: "tieto+kone" (lit. "information+machine"). Now, if we handle the word "tietokonealgoritmi" as a single token, rather than as a synonym for "algoritmi", then we end up with two separate word vectors for what are essentially the same concept, which is an inefficient use of the source data. We could try to break up, or segment, such compound words into their constituent tokens – which LAS can optionally do – but it is not obvious how well that would work in case of more complex compounds such as "tietokonealgoritmi". If that were to be segmented into the three separate tokens "tieto", "kone" and "algoritmi", it would change the semantics – "computer algorithm" would become "information machine algoritm", which is not the original meaning.

I did not attempt to segment the compound words in the source material; it seems to me, from a cursory perusal of the corpora, that while compound words are common, potentially problematic compound words such as the above example are fairly rare. Word2vec, as a stochastic technique, should be able to cope with such relatively low levels of noise or nuisance words and still give good results for the more common words. Fortunately, this turns out to be the case, as we will see.

3.2 Word2vec parameters and model training

The word2vec model involves various hyperparameters as detailed in chapter 2. As usual, I came up with suitable values of these by experimentation, given the material. The final values, as well as the key characteristics of the corpora, are listed in table 3.6. The sizes are reported per year, as a range across all the years in both corpora. Note that the averages are reported across both corpora together.

Description	Hyperparameter	Value	
Embedding dimensionality	D	100	
Context window size	C	5	
Minimum word occurrences	M	3	
Negative sampling factor	K	5	
Training epochs		30	
Data size	Parameter	Value	Average
Vocabulary size	N	120,000 – 220,000	171,000
Number of articles		55,000 – 130,000	101,000
Number of sentences		772,000 – 2,096,000	1,408,000
Number of words	Q	9,430,000 – 23,800,000	17,640,000

Table 3.6: Hyperparameters and data sizes. The data sizes are reported per year of material, across both corpora.

3.2.1 Evaluation of word2vec models

What follows is a more detailed explanation of how the hyperparameters presented before were chosen. The standard method, which I followed, is of course to try various values for each hyperparameter, i.e. to train multiple models, one per hyperparameter combination, and see which model performs best. However, in the case of word2vec, measuring model performance is not completely straightforward. In order to know which hyperparameters are the "best", we must first consider what exactly the word2vec model is doing and how its results can be reasonably verified and evaluated.

Verification during training

One performance measure that is immediately available is the value of the loss function during training. Usually the average loss over the training set is used. The loss function's value by itself, however, does not and cannot tell us how suitable the final model will be for our purposes. It is more useful as a technical indicator to verify that the model training is proceeding as expected, that is, in the direction of smaller loss, and to alert us in case of any unintented behaviour such as overtfitting. An example progression of loss is depicted in figure 3.7. I selected arbitrarily the year 2015 data set from the Yle corpus and plotted its training loss as a function of training epoch. All other years in each corpus produce a similar picture. This looks as expected, but does not yet tell us much about the quality of the resulting model.

Another thing one can keep an eye on during training is the magnitude of the embedding vectors. As is seen from the structure of the model, the expected behaviour is that these vectors should continue to grow in magnitude as training proceeds, but in such a way that the gap between smallest and largest magnitudes

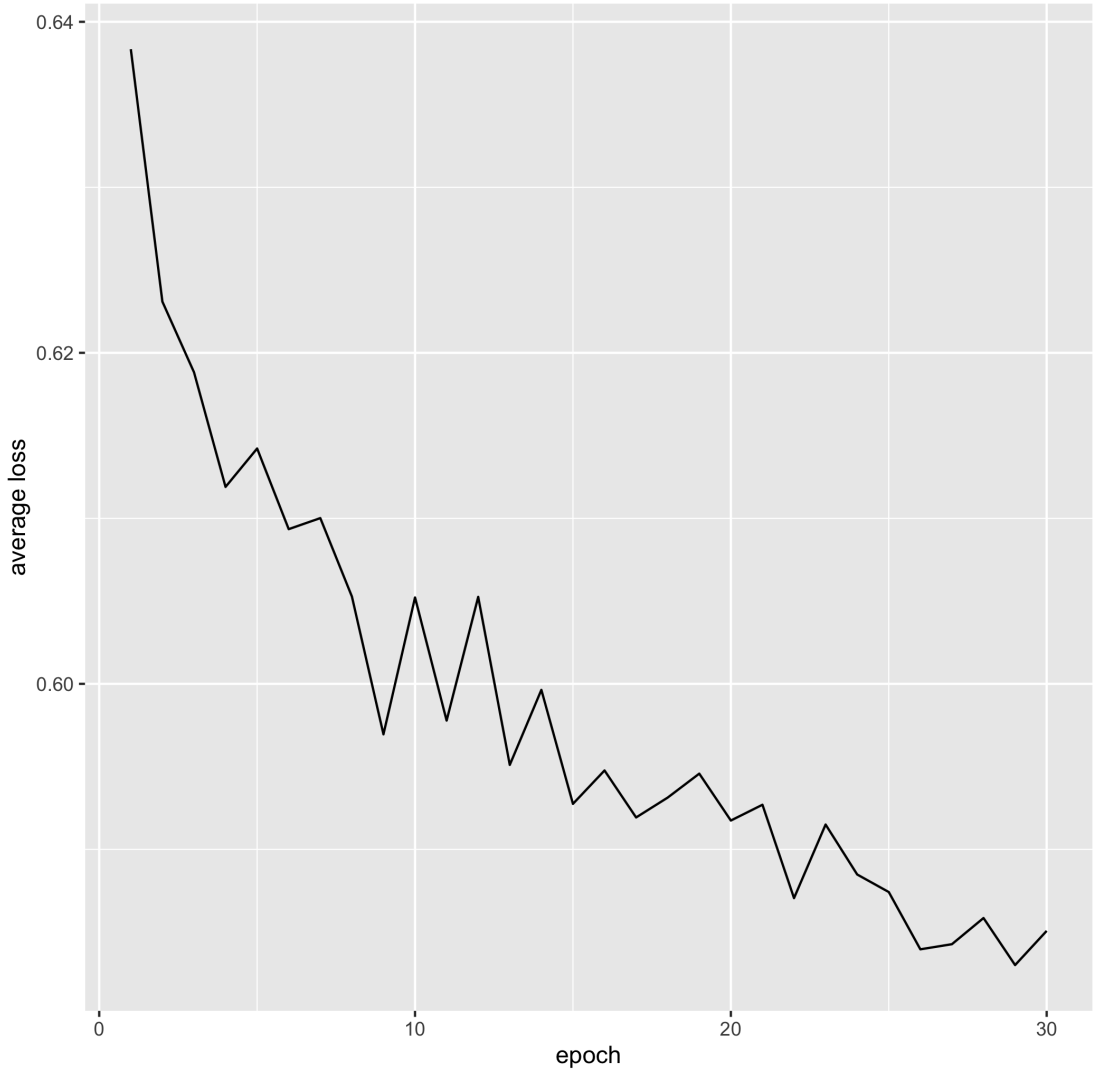

Figure 3.7: Average loss during word2vec model training. Depicted: 2015 data from Yle corpus, a typical case.

remains noticeable. The model attempts to increase the dot product between each pair of positive examples, which causes the absolute magnitude of the vectors involved to grow. At the same time, however, negative sampling causes the magnitudes of the vectors corresponding to the most common words to shrink, which, for these vectors, mitigates this growth in magnitude. It is straightforward to verify that this indeed happens as training proceeds: the magnitudes of the embedding vectors grow, and the growth seems proportional to the current learning rate as expected (gensim's word2vec implementation reduces the learning rate linearly as training proceeds).Again it must be noted though that the mere magnitude of the embedding vectors tells us only that the training is proceeding in the expected di-

rection, but is not informative with respect to the actual goodness of fit or model performance.

For supervised models, the standard technique of cross-validation [33] can be used to investigate the quality of a model during training. For this either loss values or other more relevant accuracy measures can be used. However, word2vec is an unsupervised method, and thus cross-validation cannot be used with it. Simply put, it does not make sense to ask what the performance of the model is on data it has not seen, since the word2vec algorithm does not in fact make predictions on its input data. As an autoencoder, word2vec's final result is not the nominal output of its last layer, for which the accuracy measures would normally be calculated, but instead the first-layer weight vectors, which are used as word embeddings. We should therefore focus on verifying the quality of these weight vectors. To be clear, for this verification it does not directly matter whether the model that produced the embeddings was trained with the entire corpus or, say, 90 % of the corpus, and the performance evaluation likewise does not depend on the remaining 10 %.

Direct numerical evaluation of the resulting embeddings

For many standard clustering methods, such as k-means, there are various techniques that can be used to check and ascertain the validity of the results. k-means [54] is a classic technique wherein one first picks the desired number of clusters, k – this can be, say, 3 or 5 or 10, or in general a small positive integer greater than one – and the given data points are then clustered into k distinct clusters. The clustering is done with a random initialisation, and it is not always clear which values of k are reasonable for a given data set; it is therefore useful to try to measure the validity of the resulting clustering. For algorithms such as k-means, silhouette coefficients [70] are one well-known validation method. These silhouette scores involve the mean distance between a data point and all other points in the same (model-assigned) cluster, as well as the mean distance between a data point and all other points in the next-nearest (model-assigned) cluster.

Word2vec however does not partition its results, the embeddings, into clusters, and therefore such traditional validation methods cannot be used with it: it is unclear how silhoutte scores would be calculated in the absence of clusters. Nevertheless, an evaluation of the "closeness" or "packed-togetherness" of the word2vec clustering can be attempted. It turns out that models trained with different dimensionality of embeddings, all other parameters and the data set being the same, end up with embeddings where the average similarity between each word and its nearest neighbour, as measured by cosine similarity, differs. More precisely, given a model with embedding vectors v_i, denote by $\hat{v}_i$ the embedding vector that is closest to v_i as measured by cosine similarity, i.e. $\hat{v}_i \coloneqq \arg\max_{j \neq i} \cos(v_i, v_j)$. We examine how the distribution of $\hat{v}_i$ varies for models trained with different dimensionality of embeddings, the source material and all other hyperparameters being the same.

For an arbitrarily selected year, namely Yle corpus 2014, this distribution of

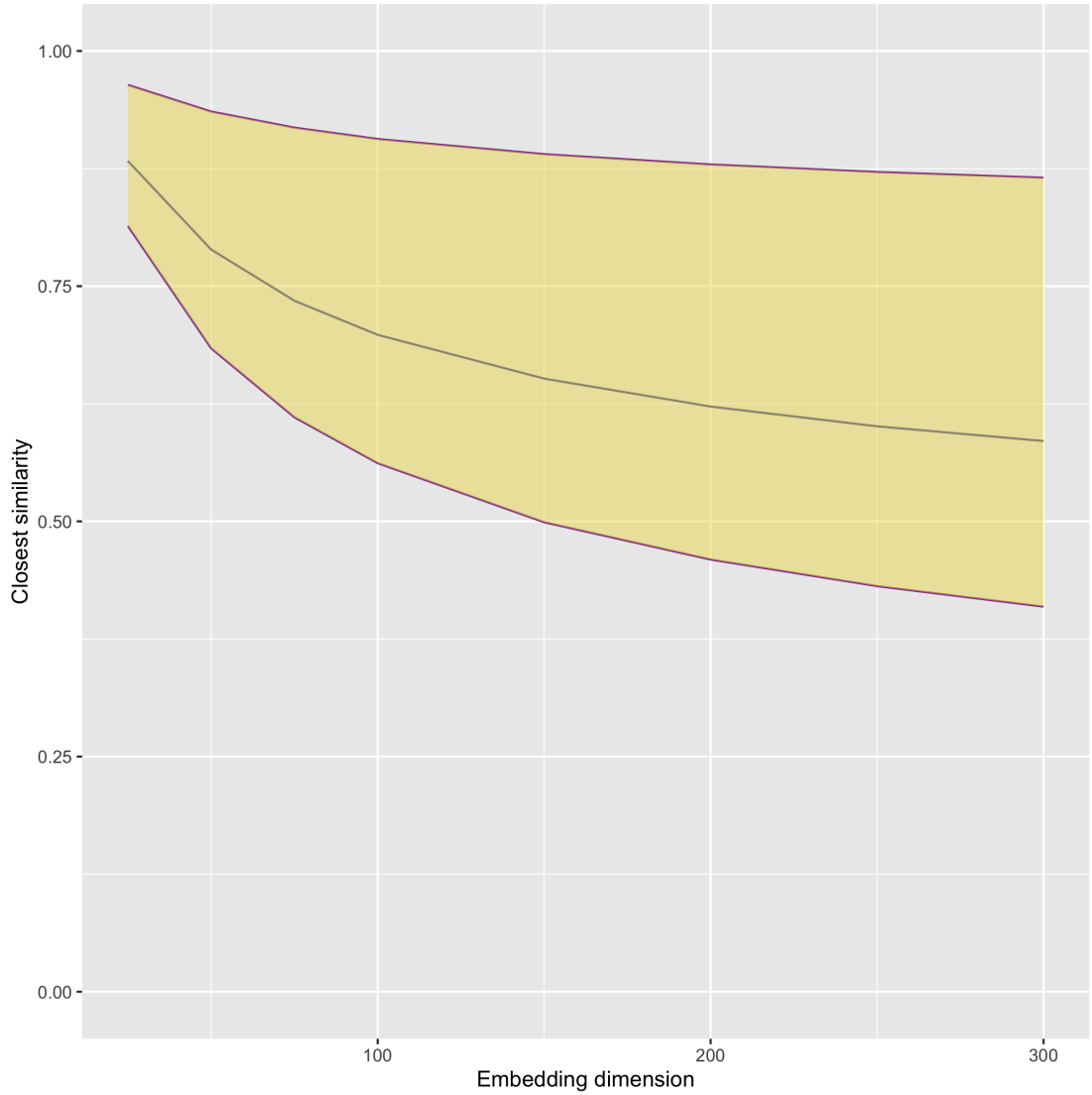

Figure 3.8: Effect of embedding dimension on the distribution of nearest-neighbour similarity. The models were trained on year 2014 of the Yle corpus.

closest-neighbour similarity by embedding dimension is depicted in figure 3.8. The average and the central 95-percentile range (i.e. from 2.5 % to 97.5 %) of $\hat{v}_i$ are shown. Note that in the word2vec implementation, the closeness function uses normalised vectors, which means that the maximum cosine similarity is 1.

It is interesting that while the 97.5th percentile of the nearest-neighbour similarity remains fairly high regardless of embedding dimension, it still decreases as the dimension grows, and the 2.5th percentile decreases very significantly, from 0.814 in the 25-dimensional case all the way to 0.409 for the 300-dimensional model. This means that the model is unable to pack the embeddings as tightly as dimensionality grows, which could indicate overfitting. However, it must be noted

that while interesting, this measure of spread does not directly inform us of how well each of the models depicted will perform on the actual task they were trained for.

Extrinsic evaluation of resulting embeddings

The main accuracy measure the original authors use is the performance of the resulting word embeddings on a word analogy task [60]. In this task, the model is asked to predict the missing word in a word analogy such as "big is to bigger as small is to X". Famously, the authors were able to use the word2vec embeddings directly as vectors to accomplish this. For example, the X in the previous could be computed via $v(X) = v(\text{small}) + v(\text{bigger}) - v(\text{big})$, where v denotes the vector corresponding to the word; the model's prediction for X would then be the word whose embedding vector is closest to $v(X)$.

In more detail, the way the analogies are used to gauge accuracy is as follows. First, for each category of analogy, a decent number of analogy pairs is written down, e.g. for the "countries and capitals" category we would have pairs such as Athens : Greece and Madrid : Spain. Then for each pair, we combine it with up to 40 other pairs to produce the quartets that are used for evaluation; this maximum number of rows with the given pair in front, 40, is here just an arbitrary constant to limit the size of the test set. Finally, each resulting quartet, such as Athens : Greece :: Madrid : Spain, is evaluated by omitting the last item, in this case Spain, and asking the model to predict what it is, in the manner described above. The model's prediction is taken to be the nearest predicted word, i.e. "top 1" instead of looking at e.g. the top 5 possibilities, and the scoring is simply 1 point for a correct answer and 0 points for the wrong answer.

Since my source material is in Finnish, the English-language analogies used by Mikolov et al. cannot be used as they are. Instead, I translated the original analogies into Finnish, with some modifications, and then used the translated analogies to attempt to gauge the performance of my models. It has to be noted that not all of the various types of analogy used by Mikolov et al. are applicable here; this is in part due to the lemmatisation, which is required due to the complexity of the Finnish language, and in part due to said complexity directly. For example, one subsection of the analogies concerns present participles such as "dance" : "dancing". During my preprocessing, all words are all lemmatised to their base forms, which makes evaluating analogies such as this impossible. But even if the words weren't lemmatised, a single Finnish translation of "dancing" does not exist, but rather, due to the way the language works, "dancing" could be translated in myriad ways, such as "tanssiminen", "tanssimassa", "tanssien" etc. Therefore this kind of analogy is not really applicable for Finnish, lemmatisation or not.

An overview of the analogies used and not used can be found in table 3.9. The entire set of analogies is described in appendix A. In the Finnish translation, as mentioned, many categories were necessarily omitted. In addition, the contents of some categories were changed; for example I modified the "common countries"

category to consist of countries physically close to Finland. I also omitted some smaller countries in the "world countries" category as well as some more obscure words in other categories. Since I was working with yearly batches of material, which are relatively small, these more obscure words and proper nouns did not occur frequently enough to justify their inclusion. I note that even if the translation were done as literally as possible, the results of the analogy test would not be comparable across languages, and as such the exact contents of the analogy test for a given language are somewhat arbitrary.

Original		Finnish translation	
:capital-common-countries			
Athens Madrid	Greece Spain	Ateena Madrid	Kreikka Espanja
:capital-world			
Amman Dublin	Jordan Ireland	Amman Dublin	Jordania Irlanti
:currency			
Denmark Russia	krone ruble	Tanska Venäjä	kruunu rupla
:city-in-state			
Chicago Denver	Illinois Colorado	-	
:family			
father he uncle	mother she aunt	isä - setä	äiti täti
:gram1-adjective-to-adverb			
calm occasional	calmly occasionally	-	
:gram2-opposite			
comfortable possible	uncomfortable impossible	mukava mahdollinen	epämukava mahdoton
:gram3-comparative			
big good	bigger better	-	
:gram4-superlative			
bad cool	worst coolest	-	
:gram5-present-participle			
dance fly	dancing flying	-	
:gram6-nationality-adjective			
Japan	Japanese	Japani	japanilainen

Sweden	Swedish	Ruotsi	ruotsalainen
:gram7-past-tense			
dancing flying	danced flew	-	
:gram8-plural			
banana child	bananas children	-	
:gram9-plural-verbs			
go listen	goes listens	-	

Table 3.9: Examples of the word pairs used in the analogy test and their Finnish translations.

Of word2vec's hyperparameters, dimensionality of the embeddings is the most important. In my testing, the other hyperparameters had much less effect on the quality of the resulting embeddings, and I therefore left them largely at the gensim implementation's defaults, as described in table 3.6. The effect of embedding dimensionality on accuracy, as measured by performance on the analogy task, is depicted in figures 3.10 and 3.11. I selected arbitrarily a handful of years from each corpus, and for each year trained a series of models with different embedding dimension, all other parameters being the same. The accuracy of these five models is shown in figure 3.10. The average accuracy across the five models, by dimension, is depicted in figure 3.11.

It appears from figure 3.10 that the model begins to overfit as dimensionality grows beyond 100–150. Overfitting, of course, is traditionally defined as related to the generalisation error of a model, i.e. a model's ability to produce consistently decent predictions when faced with new data [43, 15]. As such, the concept of overfitting does not completely apply to word2vec, since the word2vec model is not used for predictions. However, overfitting refers more generally to a situation where a model of too high complexity is used, such that the model can perform well on the training data simply by memorising it. This can clearly occur with the word2vec model, which has a very large number of parameters, regardless of whether said model is then used for predictions or not. The non-predictive nature of the model, then, merely makes it more tricky to ascertain whether overfitting has occurred.

A more detailed breakdown of model performance on the various subtasks of the analogy test, by embedding dimensionality, is shown in figure 3.12. The source data for the models depicted here is year 2011 of the Yle corpus; the picture is very similar for the other years I checked across both corpora. It is interesting to note that, for Finnish text, the opposites task is clearly the most difficult, and in general there are wide differences in model performance on the various subtasks.

In the original word2vec paper [58] the authors do not report their accuracy results quite this granularly. One of their models (a skip-gram) achieves a total

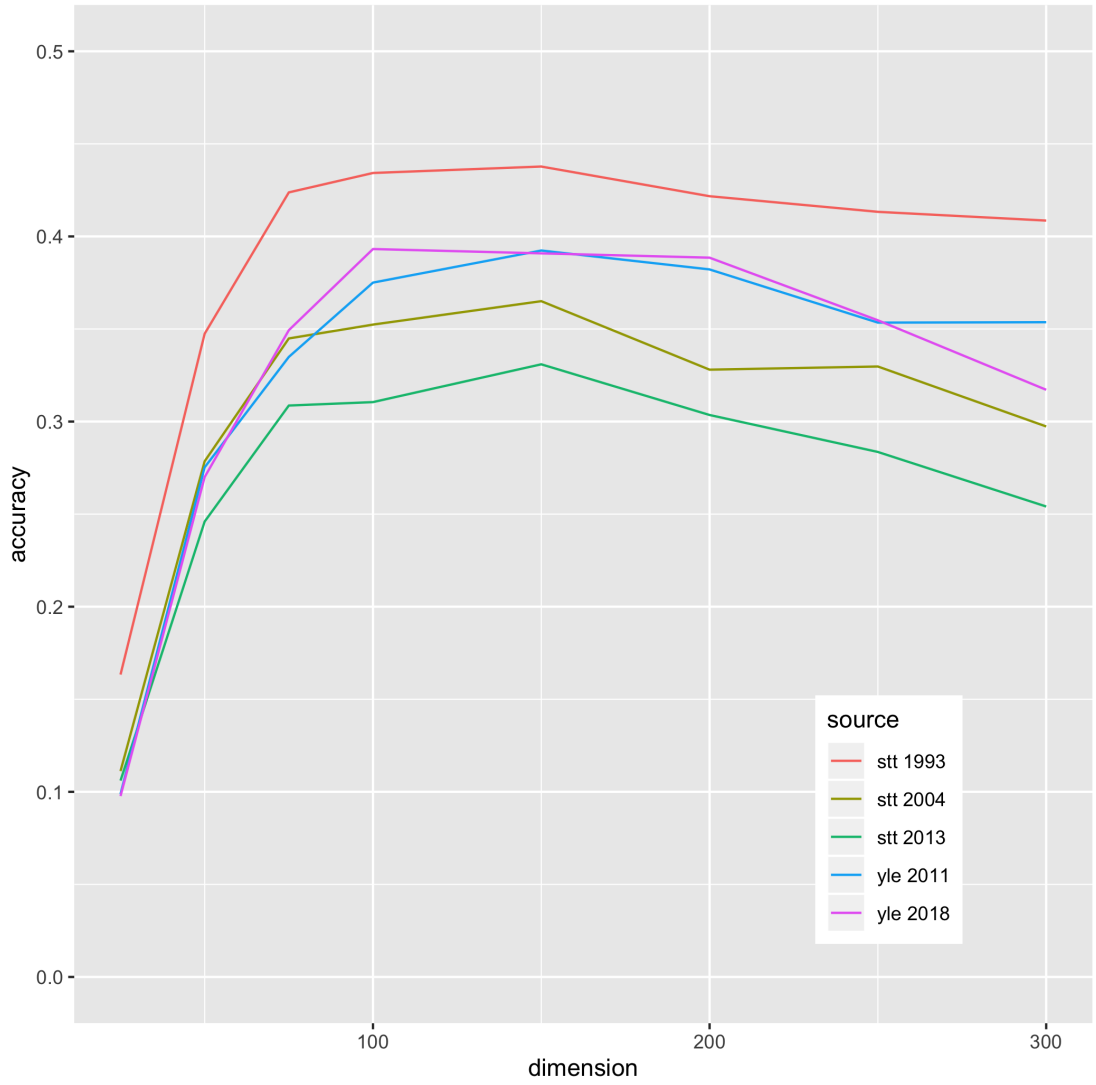

Figure 3.10: Accuracy on analogy task by embedding dimension: five arbitrarily selected data sets.

accuracy of 53.3 % on their analogy task; this was for a 300-dimensional model trained on 783 million words. Another skip-gram model was trained on 6 billion words and achieved a total accuracy of 65.6 %. In comparison, the best average accuracy across the five models, as shown in figure 3.11, was 38.3 %; the word counts of the source material for these five models were between 13.2 million and 21.1 million. It is fair to say, therefore, that word2vec models can achieve decent accuracy on the analogy task even with relatively low word counts.

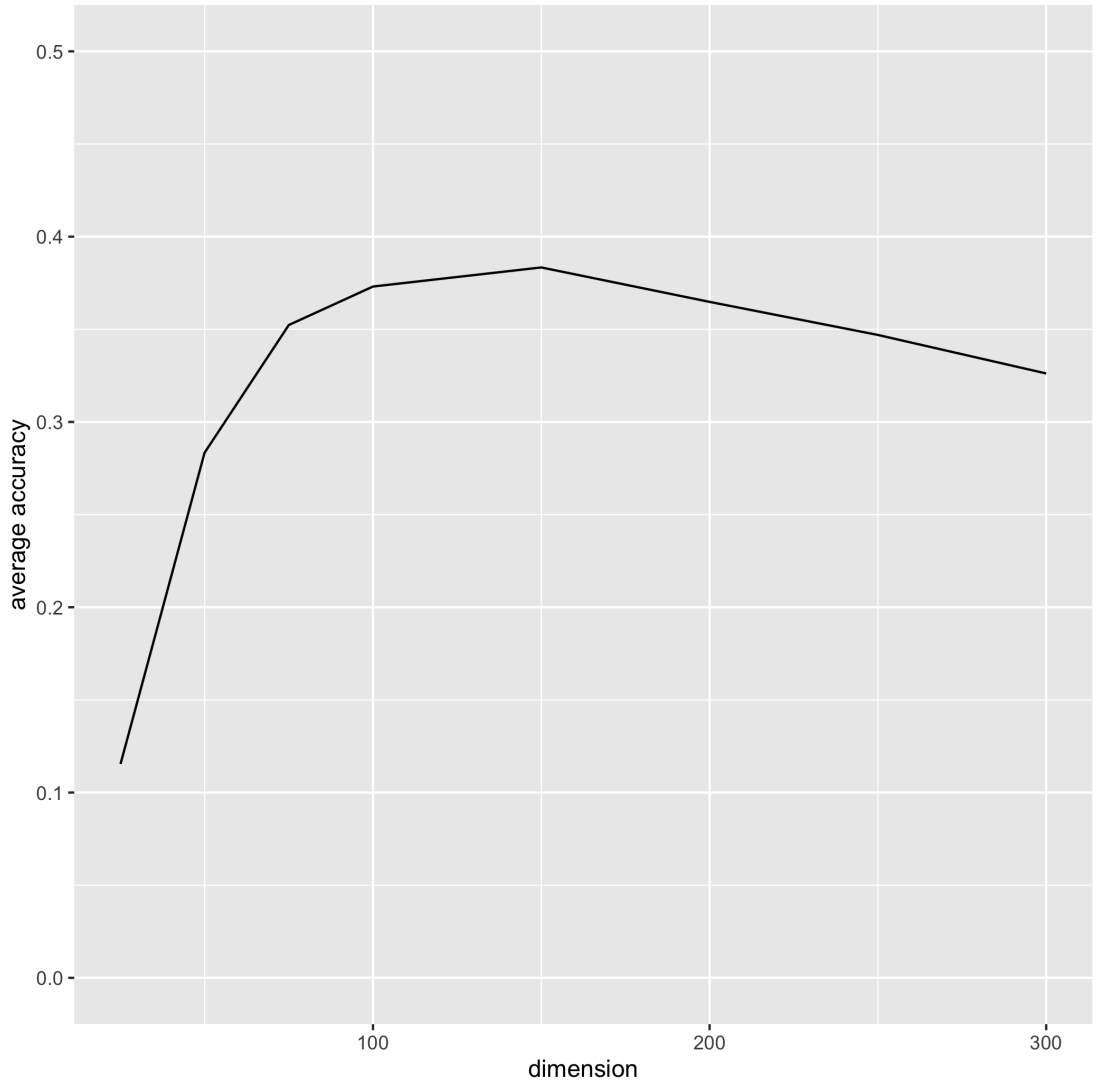

Figure 3.11: Accuracy on analogy task by embedding dimension: average across the five data sets.

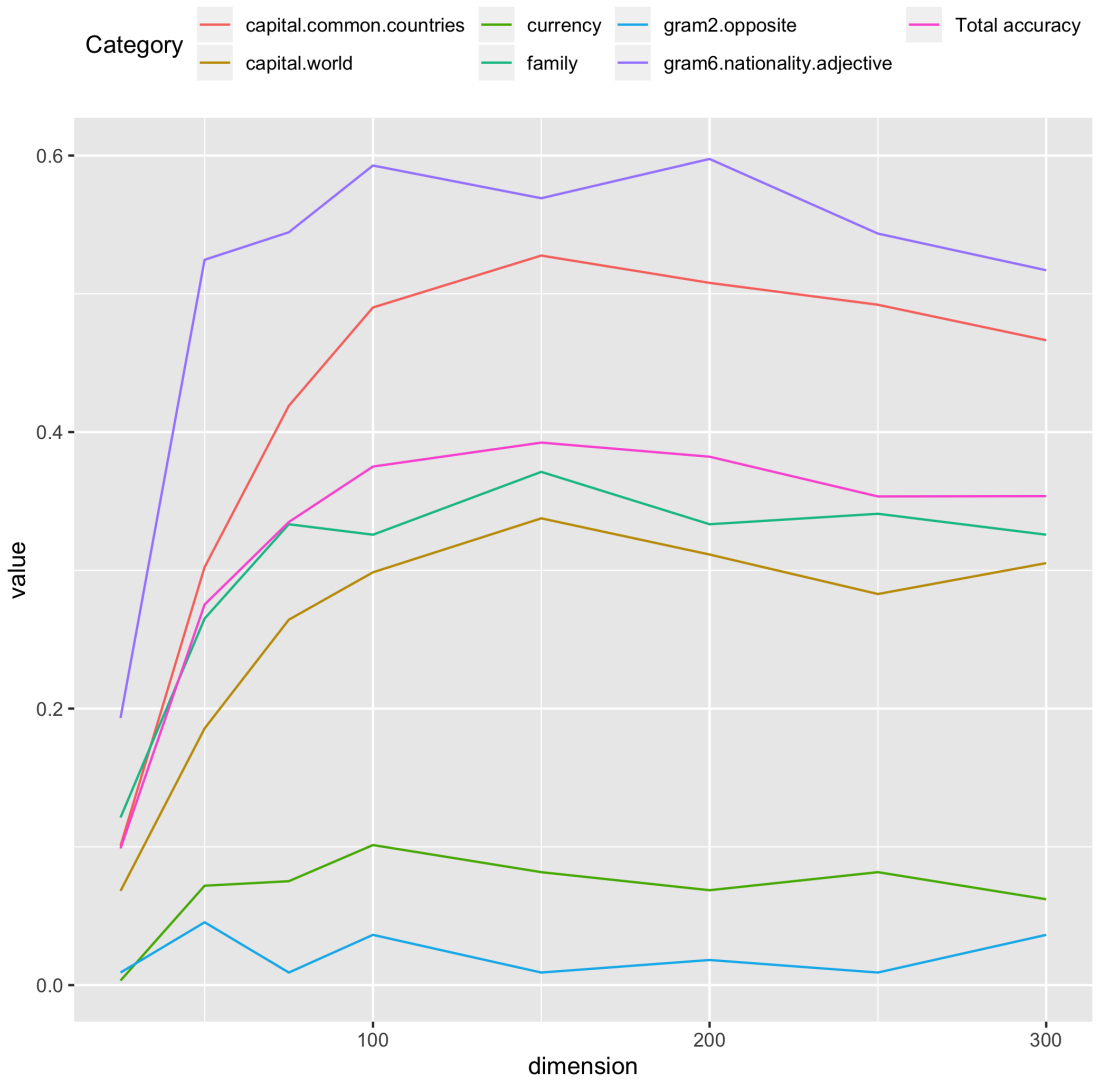

Figure 3.12: Accuracy on analogy task by embedding dimension, broken down by subtask. The total accuracy is plotted in purple. Source material: Yle corpus, year 2011; a representative case.

Chapter 4

Experimental results

One direct application of word embeddings as produced by word2vec is to examine the changes in word meanings over time. This idea has been investigated by various authors [23, 32, 45, 47, 81]; a survey is provided by [48]. Such a study of word semantics over time can be done by training a series of word2vec models, e.g. one per year of source material, and then comparing the change in the nearest neighbours of each word of interest over time. With word2vec embeddings, such comparisons can be made in a measurable way, as we will see.

In what follows I present first the results of an analysis made on the basis of the Yle corpus (years 2011–2018). After this I repeat the analysis for the STT corpus (years 1997–2018 only).

I chose as my topic the current public discussion on artificial intelligence. This discussion encompasses many related concepts, such smart algorithms, machine learning, intelligent robotics and so on. It is well known, and easily noticeable, that these topics have been receiving increasing amounts of coverage in mainstream media in the last few years [20, 67]. It can therefore be reasonably hypothesised that perhaps the context of certain words related to thse topics has changed as well. Starting from Firth's famous remark that "you shall know a word by the company it keeps" [21], i.e. that the meaning of a word is in its context, we can examine the contexts of various words of interest using word2vec, which computes approximations of these contexts in the form of word embeddings. Furthermore, the changes in these contexts – and therefore in the meanings of the words – can be examined by observing changes in word2vec models trained from the words. This can be done by training several word2vec models, e.g. one for each year of material, and observing the embeddings of selected words in various models.

To be more concise, the hypothesis can be stated as follows.

> **Hypothesis.** Assume that the meaning of a certain concept has changed over time. Then this change in meaning is reflected in changes in word contexts, for those words that are most closely related to said concept. Furthermore, with a computational method such as word2vec, one can attempt to approximate and quantify this change in word contexts, in order to better understand the change in meaning.

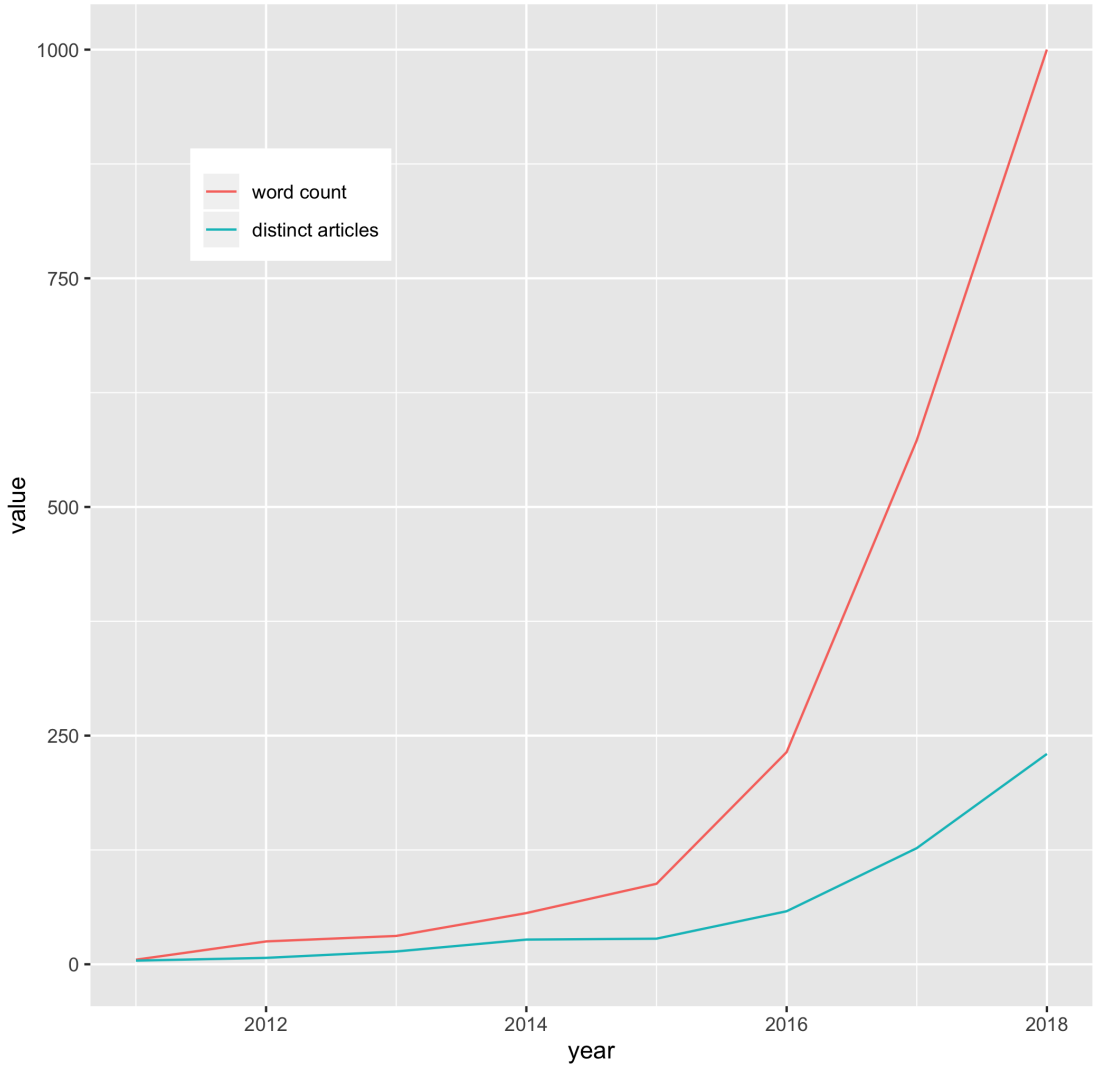

Figure 4.1: Occurrences of the word "tekoäly" (artificial intelligence) in the Yle corpus. Both overall word counts and the number of distinct articles containing the word are depicted.

With all this in mind, I had a look at the word "tekoäly" (artificial intelligence) and how its nearest neighbours change over time. The frequencies of "tekoäly" in the Yle corpus between 2011 and 2018 is depicted in picture 4.1. The fact that in Finnish "tekoäly" is a compound word, i.e. a single word, makes the analysis somewhat more convenient; when working with phrases such as the English "artificial intelligence", one would have to first preprocess the material in such a way that such multi-word phrases are treated as one token, but in my case this was not necessary.

As is seen in figure 4.1, the number of occurrences of the word "tekoäly" in

the Yle corpus has grown exponentially in the few years up to 2018, whether one measures by overall word count or by the number of distinct articles containing the word. In 2011, there were 4 articles and a total of 5 mentions of "tekoäly", while in 2018 these had risen to 230 and a round 1000 respectively. Based on this we can reasonably expect there to have also been a measure of change in the contexts in which the word is used, and it is this that we analyse next.

4.1 Method

In more detail, the method for analysing the target word, visualising the results, and discovering new, related target words for further analysis is as follows. This procedure is adapted from [32].

1. Find out appropriate values for the various word2vec hyperparameters through exploration (see section 3.2).

2. With appropriate hyperparameter values, train a word2vec model with the first year's material from the chosen corpus.

3. Advance to the next year of material and retrain the previous word2vec model with it; i.e. use the current word embeddings as the starting point, and expand the vocabulary with any new words, initialising the corresponding new word embeddings randomly.

4. Repeat the previous step for each year of material.

5. Select the years of interest for examining the word embeddings. For example, for the Yle corpus I chose the years 2011, 2014 and 2018 (the first year, a middle year and the last year respectively).

6. Examine the k nearest neighbours of the target word, for each year of interest (k can be, say, 10 or 20), and compare these sets of neighbours between years. For example, for the target word "tekoäly" in the Yle corpus, some of these yearly comparisons are depicted in detail in tables 4.2 and 4.3 (for details, see below).

7. For visualisation, there are two main steps: visualising the current (i.e. latest-model) positions of the target word and each of its neighbouring words, and visualising the older (i.e. earlier-model) positions of the target word. There are various ways to optimise and tweak the visualisation to obtain better results. Here, the overall method is presented, and the details of the tweaks are described below.

8. First, the target word being t, let $n(t)$ denote the set of all neighbouring words that we selected for visualisation across all years of interest. We select the vectors corresponding to the words $n(t)$ from the newest model, as the

purpose is to display the latest position of the neighbouring words. Next, we fit a dimensionality reduction model on these vectors. Suitable dimensionality reduction techniques include t-SNE [53] and the newer UMAP [56]; if desired, the standard preprocessing step of principal component analysis (PCA) can be used as well. Call the fitted dimensionality reduction model F. We can now use F to compute the two-dimensional embedding of the neighbours n or any subset thereof. We also pick the newest model's word vector for the target word t and compute its two-dimensional embedding with F.

9. Next, we visualise the position of the target word according to the earlier models. To do this, simply read the vector corresponding to the target word from each model corresponding to an earlier year of interest. These older target vectors can now also be two-dimensionalised with F.

10. After computing the two-dimensional representation of the desired word vectors, all that remains is drawing them. In some scenarios it may be desirable to compute the dimensionality reduction for a larger amount of word vectors than are actually required, in which case one would now draw only the desired vectors rather than all of them. For details, see below.

Some notes on the visualisation procedure follow. First of all, there are two ways to choose which, or how many, neighbouring words to consider for visualisation. It can be useful to utilise a larger amount of neighbours for computing the visualisation, as this computation is done with a dimensionality reduction method that can benefit from the added stability provided by a relatively large number of word vectors; one can then choose, for clarity, to display only some of these two-dimensionalised word vectors.

For example, for the word "tekoäly" in the Yle corpus, I experimented with values of k between 20 and 100 for computing the visualisation, ending up with $k = 50$ nearest neighbours per year; that is, I chose the top $k = 50$ neighbours from 2011, another $k = 50$ neighbours from 2014 and $k = 50$ neighbours from 2018, and computed the two-dimensional visualisation based on these 150 vectors. (Note that there is some overlap in the nearest neighbours; this is benign.) After the computation, I then drew only the nearest 10 neighbours per year, and from these I selected 4 per year for the final image.

One can also select a smaller number of neighbours per year to begin with, say 3 or 4 out of the top 10, and compute the dimensionality reduction directly for this smaller set. Either method is valid, as the visualised words are in either case picked from the same models. The potential issue with computing the two-dimensional reduction from a smaller number of word vectors is that the results may not be satisfactory: it can happen that the dimensionality reduction places the two-dimensional images of the vectors in the "wrong order" from the point of view of their original cosine similarities, e.g. in "2-1-3" order rather than "1-2-3" or "3-2-1". This can often be mitigated by providing a larger amount of original vectors to

the dimensionality reduction algorithm as described above, so that the algorithm has a better chance of figuring out the regularities in the cosine similarities of the data. In practice, one might have to try both of the methods described here and perhaps also tweak various method-specific parameters to obtain good results, as dimensionality reduction is an inherently tricky problem.

Another issue to be considered is the alignment of the word vectors across years. While word2vec embeddings are of course consistent with one another within each yearly corpus, there is nothing that guarantees similar alignment of a given word's embedding vector between two different models, such as the yearly-trained models in this scheme. In the procedure described above, each year's model is trained starting from the previous year's model, so the vectors – or at least, those whose words are shared between years – do retain at least some of their orientation from year to year. However, the vectors may still drift out of alignment over time, especially if the number of years and therefore training rounds is very large.

To better align the vectors, an orthogonal Procrustes transformation [72] can be used. With this transformation, we find the rotation matrix that rotates model B's vectors so that they are aligned, as closely as possible, with model A's vectors, in such a way that the between-column alignment within either embedding matrix is preserved.

There are two details about this procedure that warrant mentioning. First of all, exactly how the Procrustes alignment fits in with the rest of the model training depends on the word2vec implementation. In the gensim implementation that I used, it is not straightforward to manipulate the model vectors in such a way that the modified vectors can then be used to train the next model. In practice, this simply means that the models must be trained in sequence with no Procrustes alignments, and at the end one can then perform an additional step of aligning all the trained models with one another in sequence. This is straightforward, but care must be taken to use the rotated model vectors afterwards, as these will be in a different format than the ordinary gensim models.

More importantly, Procrustes alignment is not straightforward to apply in case some words (and therefore word vectors) are not present in every model. The standard way of computing the orthogonal Procrustes alignment assumes that each of the two matrices being considered has the same number of columns and that the columns are in the same order. However, as an example, some of the nearest neighbours of "tekoäly" in 2018 are not present in all previous models: "koneoppiminen" (machine learning), for instance, first appears in 2014 in the Yle corpus, while said corpus goes back to 2011. In principle, it could be possible to create an extrapolated vector for such words for the years when they were not yet present, but it is not immediately clear how valid the results of this would be. The easiest solution to this dilemma is to use the original unaligned vectors, rather than the Procrustes-aligned ones, for any words that are not present throughout the timeline of the corpus.

In practice, one must be ready to experiment with various solutions, both with Procrustes alignment and without, as well as with different parameters of

the underlying dimensionality reduction technique, in order to find a passable visualisation.

4.2 Results: Yle corpus

The basis of the analysis is a straightforward comparison of the nearest neighbours of the target word, as computed by word2vec models trained on separate years of material. For the Yle corpus, I chose the years 2011, 2014 and 2018 as the years of interest; these are the first, middle and last years of the corpus. These nearest neighbours are then easy to read out directly once the models have been trained. To begin with, this has been done for the first two years, 2011 and 2014, in table 4.2 for the target word "tekoäly". Notice that some of the words have been altered by the lemmatisation: some compound words have had dashes added, while other compound words are no longer grammatically correct, such as the lemmatised word "koneoppiminenmenetelmä". In addition, the source material's phrase "South by Southwest" (the name of a music and film festival) has resulted in the single, wrongly lemmatised token "southvesti". These minor issues do not affect the results noticeably.

From table 4.2 it is seen that the context of the word "tekoäly", as computed by word2vec, changes somewhat between 2011 and 2014. In 2011 the word "tekoäly" referred to a variety of technological concepts, such as weapons technology and environmental and computer technology, as well as technical applications such as quizzes. In contrast, while in 2014 "tekoäly" is still strongly associated with various practical applications (Hong Kong metro system as implemented by MTR, facial recognition, mobile tickets), it has also come to be associated with more abstract computer science concepts such as algorithms, bots and computer programs in general. This change in context can also be verified by checking where the 2014 model places the words that were closest to "tekoäly" in the 2011 model, which is done in the last segment of table 4.2. We see that the top ten words from 2011 are now fairly distant in 2014: all of them are significantly further away than the top ten words from 2014, and some, such as weapons technology, have ended up very far in the 2014 model.

Table 4.3 shows the ten words nearest to "tekoäly" in the model constructed from the year 2018 data, along with a look at 2011's nearest words from the 2018 model. We can see that the change in context, and therefore arguably the meaning, is even more marked from 2011 to 2018 than it was between 2011 and 2014. In 2018, the word "tekoäly" is most closely associated with machine learning, deep learning and algorithms, i.e. specific computer science concepts. An interesting facet of this change in context is that the nearest neighbours in 2018 no longer include any specific applications, but are rather fairly generic, broad terms. Looking at the words that were the nearest neighbours in 2011, we see that in 2018 they are again more distant than 2018's top ten words. Roughly half of these 2011 words have moved farther away since 2014, while the other half has moved slightly closer,

Neighbouring word	Distance	English description
tekoäly (2011)		
tietovisailu	0.6703	quiz; game show
uutinenrintama	0.6248	news front
dcnsn	0.6127	genitive of DCNS (defence company)
androidkäyttöjärjestelmä	0.6021	Android operating system
aseteollisuus	0.6011	weapons industry
ympäristöteknologi	0.5982	environmental technology
kehitystavoite	0.5948	development goal
tietokonevalmistaja	0.5893	computer manufacturer
teknologi	0.5864	technology
aseteknologia	0.5855	weapons technology
tekoäly (2014)		
mtrn	0.6954	MTR (Hong Kong transport company)
algoritmi	0.6165	algorithm
tietokoneohjelma	0.6127	computer program
botti	0.6085	bot (i.e. "software robot")
kasvontunnistus	0.5954	face recognition
mobiililippu	0.595	mobile ticket
southvesti	0.59	South by Southwest (festival)
ohjelmoida	0.589	to program (verb)
cortanasovellus	0.5889	Cortana application
icf	0.5887	Intelligent Community Forum
tekoäly (2011 words in 2014 model)		
tietovisailu	0.4436	quiz; game show
uutinenrintama	0.3574	news front
dcnsn	0.4184	genitive of DCNS (defence company)
androidkäyttöjärjestelmä	0.4234	Android operating system
aseteollisuus	0.1979	weapons industry
ympäristöteknologi	0.3481	environmental technology
kehitystavoite	0.3739	development goal
tietokonevalmistaja	0.5097	computer manufacturer
teknologi	0.4362	technology
aseteknologia	0.2844	weapons technology

Table 4.2: Nearest neighbours of "tekoäly" (artificial intelligence) in the Yle corpus, years 2011 & 2014.

but no words have ended up near the top ten.

All in all, these findings support the hypothesis that the usage, and therefore arguably the meaning, of the word "tekoäly" has changed between 2011 and 2018. We see that not only have the nearest words changed noticeably over time, but also the category of these nearest neighbours has changed: in 2011, "tekoäly" occurred

Neighbouring word	Distance	English description
tekoäly (2018)		
koneoppiminen	0.7916	machine learning
teknologia	0.7823	technology
koneoppiminenmenetelmä	0.7471	machine learning method
algoritmi	0.7468	algorithm
teknologinen	0.729	technological
syväoppiminen	0.7214	deep learning
super-tekoäly	0.7114	super AI (artificial intelligence)
robotti	0.7101	robot
keino-äly	0.7077	constructed intelligence
digitaaliteknologia	0.7048	digital technology
tekoäly (2011 words in 2018 model)		
tietovisailu	0.4287	quiz; game show
uutinenrintama	0.3027	news front
dcnsn	0.3712	genitive of DCNS (defence company)
androidkäyttöjärjestelmä	0.5605	Android operating system
aseteollisuus	0.3825	weapons industry
ympäristöteknologi	0.5254	environmental technology
kehitystavoite	0.3806	development goal
tietokonevalmistaja	0.5181	computer manufacturer
teknologi	0.6013	technology
aseteknologia	0.3609	weapons technology

Table 4.3: Nearest neighbours of "tekoäly" (artificial intelligence) in the Yle corpus, years 2011 & 2018.

most often near words which refer to specific applications, while in 2018 the word is most closely associated with broader, more abstract concepts.

Using the method described in the previous section, this change in nearest neighbours between 2011 and 2018 can also be visualised, which is done in picture 4.4. For clarity, I have chosen to include only some of the nearest neighbours for each of the years 2011, 2014 and 2018. The red dots depict the words that were among the $k = 10$ nearest neighbours in 2011, 2014 or 2018. The blue arrow shows the "journey" of the word "tekoäly" itself between 2011 and 2018. This visualisation is, of course, just a different way of describing some of the changes that were detailed in tables 4.2 and 4.3. For this particular picture, I used UMAP [56] for the dimensionality reduction and did not need to use either PCA or Procrustes alignment for preprocessing.

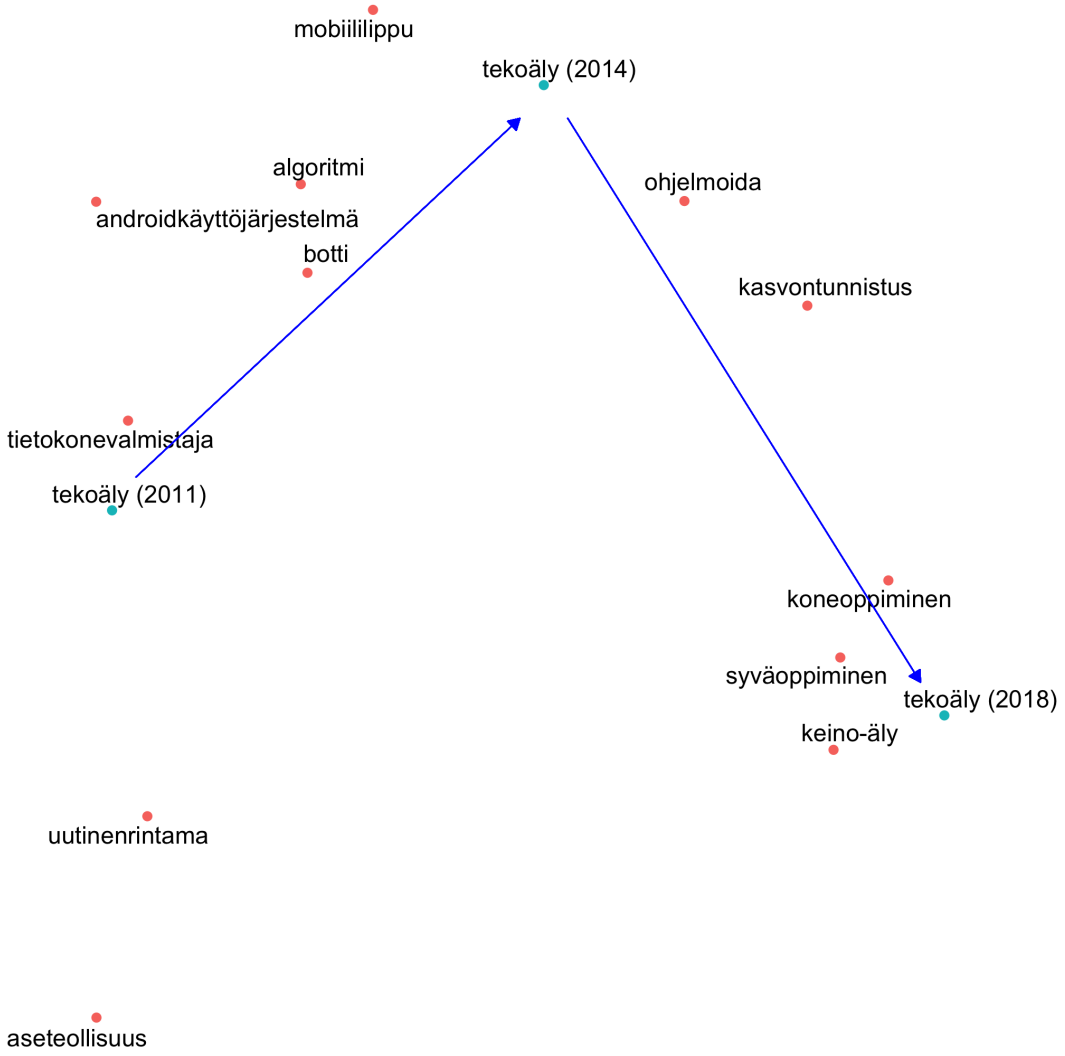

Figure 4.4: A comparison of the nearest neighbours of the word "tekoäly" (artificial intelligence) between 2011 and 2018, Yle corpus. The red points indicate those words that were among "tekoäly"'s nearest neighbours in 2011, 2014 or 2018.

4.2.1 Other interesting words

The tables 4.2 and 4.3 lists the nearest neighbours of our target word, "tekoäly", and one avenue of further analysis is picking out some of these words for a closer look. For example, we can see that the words "algoritmi" (algorithm) and "robotti" (robot) are only close to "tekoäly" in the year 2018 model. In fact, digging a bit deeper reveals that for the 2011 model, the set of the $k = 1000$ words nearest "tekoäly" does not include either "algoritmi" or "robotti". This is interesting, since both of these words are in the top ten neighbours of "tekoäly" for 2018. Further-

Neighbouring word	Distance	English description
algoritmi (2011)		
gpsvastaanotin	0.6818	GPS receiver
ip-osoite	0.6748	IP address
liikuntaharjoittelu	0.6734	healthy exercise
alaryhmä	0.6715	subgroup
androidkäyttöjärjestelmä	0.6666	Android operating system
naturelehti	0.6658	the journal Nature
soluviljelytutkimus	0.6612	cell culture research
päättelykyky	0.6606	inference capability
nyky-teknologia	0.6582	modern technology
moni-mutkaistuminen	0.658	increasing complexity
algoritmi (2018)		
tekoäly	0.7468	artificial intelligence
gsa	0.6985	GSA (EU satellite navigation agency)
kryptografia	0.6866	cryptography
koneoppiminen	0.6864	machine learning
somemiljardööri	0.678	social media billionaire
tweetdeck	0.6739	tweetdeck
koneoppiminenmenetelmä	0.6737	machine learning method
hakukonepalvelu	0.6719	search engine service
google	0.6716	Google
gesture	0.6696	gesture

Table 4.5: Nearest neighbours of "algoritmi" (algorithm) in the Yle corpus.

more, the word "robotti" seems like something that could reasonably be close to the apparent meaning of "tekoäly" in 2011, which included weapons, environmental and computer technology.

To gain more insight into word contexts, it is easy to repeat the analysis described above for other words of interest, which I have done in the following for the words "algoritmi" and "robotti". Notice that in the following tables, some of the words have again been rendered grammatically incorrect by lemmatisation: some dashes are added, and most dramatically, a certain inflection of the Finnish word "painoton" ("weightless") has been mis-lemmatised into the incorrect "paiot-tomassa" (correct: "painottomassa"). In addition, the foreign names Costigan and Tynker have been incorrectly lemmatised. As before, these issues do not affect the results.

As can be seen from table 4.5, in 2011 the word "algoritmi" was associated with various technical and scientific concepts: GPS receiver, Android operating system, biological research, etc. In 2018, "algoritmi" is again closely related to various applications such as cryptography and search engines, but it is also now close to artificial intelligence and machine learning. In other words, although the

Neighbouring word	Distance	English description
robotti (2011)		
petman	0.6707	PETMAN (name of a robot)
vanhuskeskus	0.6368	senior citizen centre
robottihylje	0.6071	robot seal
paiottomassa	0.6058	in a weightless state
ihmiskäsi	0.5952	human hand
etä-käyttö	0.5789	remote usage
sikiödiagnostiikkayksikkö	0.572	fetus diagnostics unit
kuusijalkainen	0.5718	six-legged
costiga	0.5717	Costigan (a name)
tietokonetomografia	0.5593	computer tomography
robotti (2018)		
tekoäly	0.7101	artificial intelligence
hirukawa	0.6948	Hirukawa (name)
teknologia	0.6763	technology
etä-kirurgia	0.6558	remote surgery
tuntonäyttö	0.6543	touch display
cyberdyne	0.6432	Cyberdyne (company)
tynkeri	0.642	Tynker (game)
teknoasia	0.6411	techno[logical] issue
koneoppiminen	0.6379	machine learning
gulzar	0.6373	Gulzar (name)

Table 4.6: Nearest neighbours of "robotti" (robot) in the Yle corpus.

meaning has not completely changed based on this analysis – "algoritmi" is still closely associated with a collection of various practical applications – we now, in 2018, find it also to be close to AI and machine learning, where this was not the case in 2011. In this respect, the change of meaning is similar to that of "tekoäly", which we looked at earlier.

In the same vein, "robotti", as is seen in table 4.6, has in this corpus always referred to fairly traditional concepts of robots, but the difference in 2018 is that it has come to be associated also with artificial intelligence and machine learning. These concepts are more abstract than physical robots, and we can therefore argue that the meaning of "robotti" has widened to include also such "conceptual robots", in addition to the traditional ones. This change of meaning is very similar to that of "algoritmi" and "tekoäly" analysed earlier.

As before, I have visualised this change in nearest neighbours between 2011 and 2018 in pictures 4.7 and 4.8 for "algoritmi" and "robotti" respectively. Again only some of the nearest neighbours for both 2011 and 2018 are depicted. The green dots indicate the words that were among the $k = 10$ nearest neighbours in 2011, and similarly the red dots depict those from 2018. The blue arrows show the

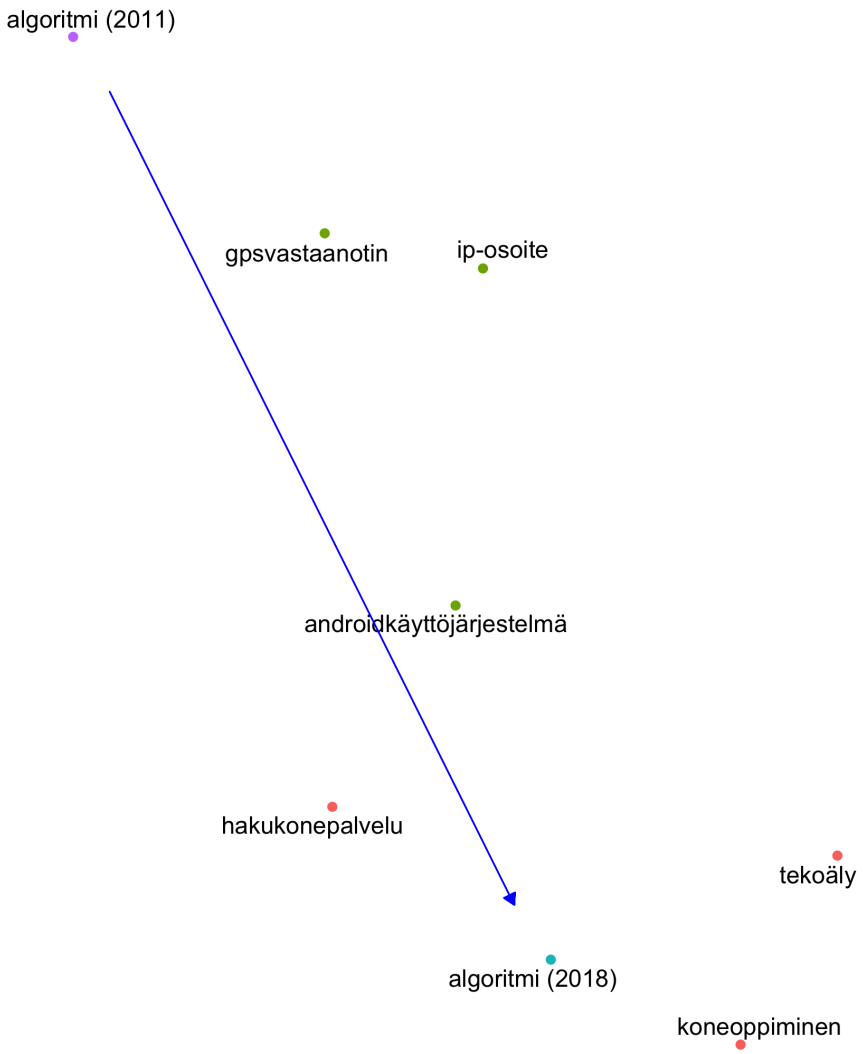

Figure 4.7: A comparison of the nearest neighbours of the word "algoritmi" (algorithm) between 2011 and 2018. Source: Yle corpus. Green and red points indicate those words that were among the nearest neighbours in 2011 and 2018 respectively.

"journey" of the target word itself between 2011 and 2018.

4.3 Results: STT corpus

As I had two distinct corpora available, Yle and STT, I repeated the analysis on the latter as well. First of all, I looked at the number of occurrences of the word "tekoäly" in the STT corpus. A graph of this is seen in figure 4.9. These occurrence counts are very similar to those of the Yle corpus, as seen in the previous figure 4.1, in that the number of mentions, whether measured by word count or the

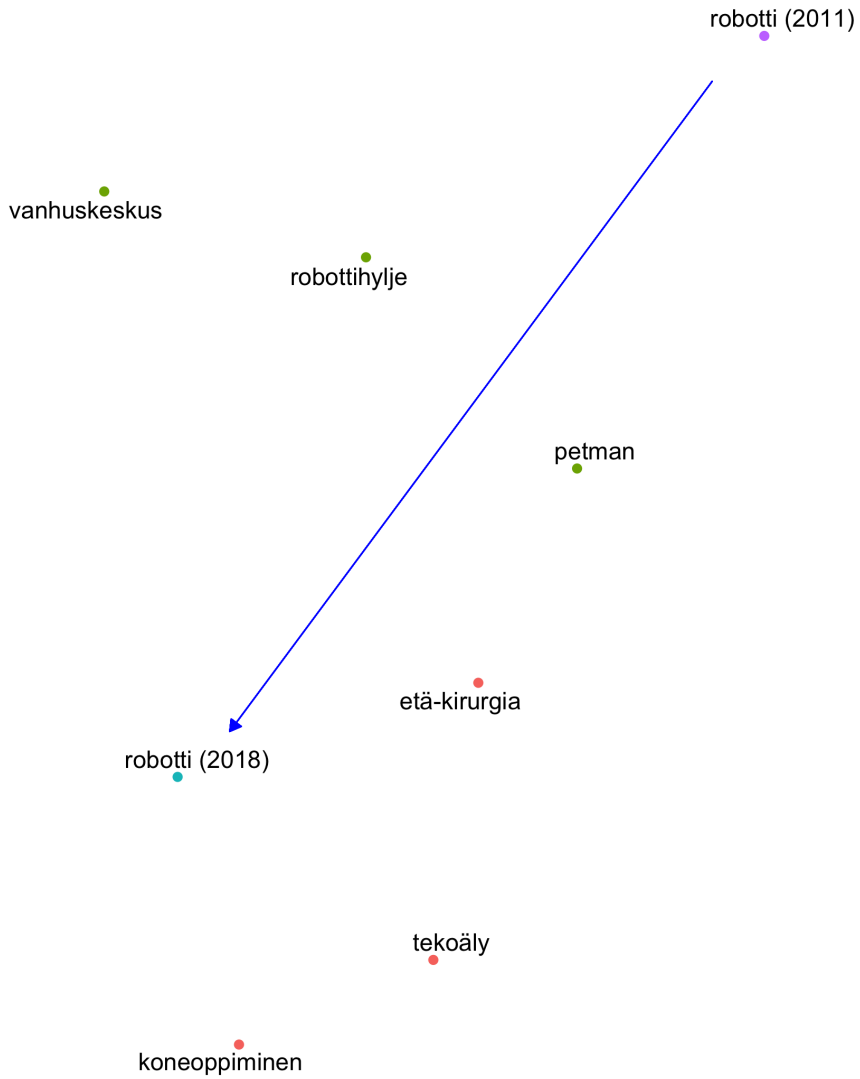

Figure 4.8: A comparison of the nearest neighbours of the word “robotti” (robot) between 2011 and 2018. Source: Yle corpus. Green and red points indicate those words that were among the nearest neighbours in 2011 and 2018 respectively.

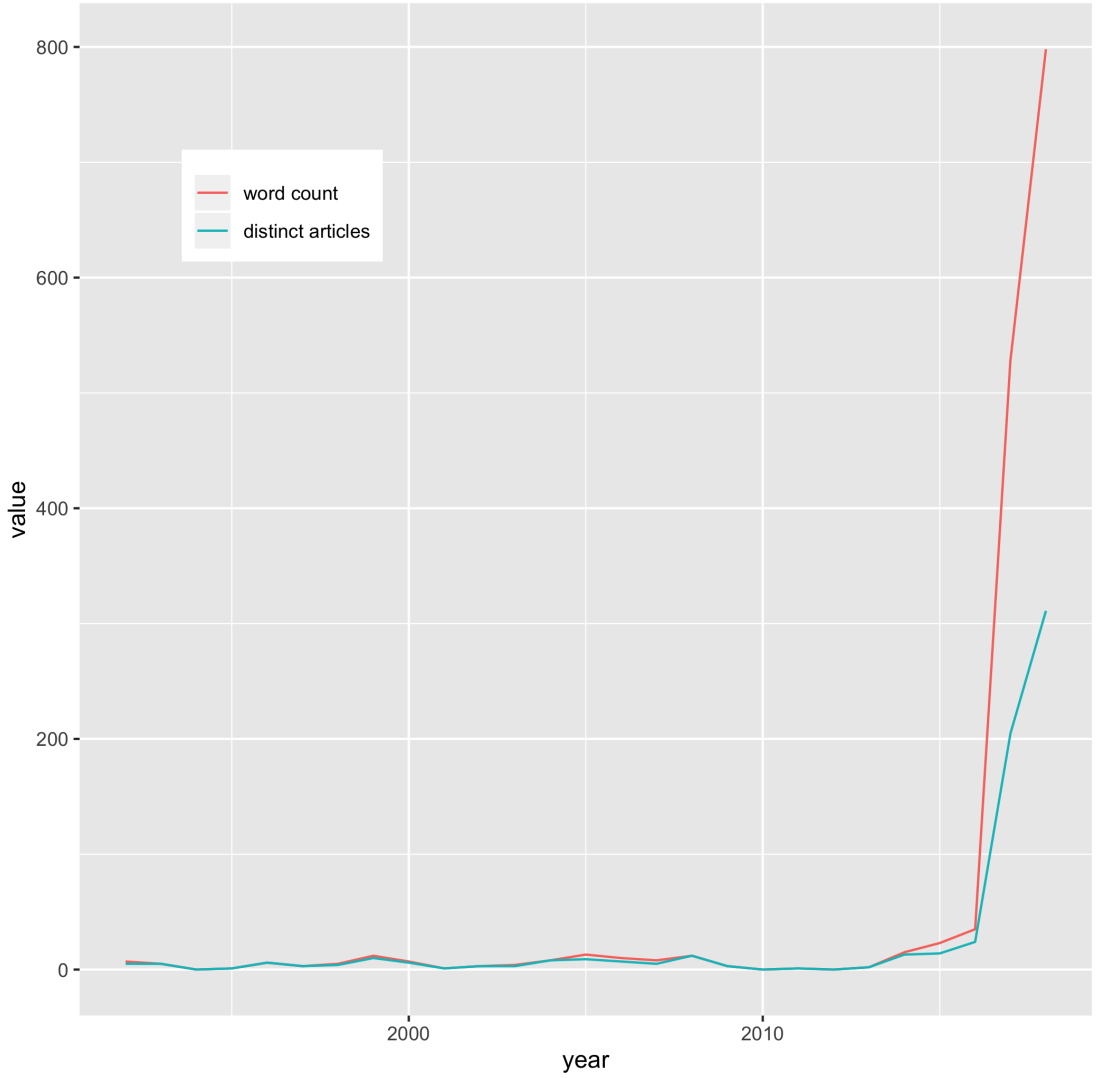

Figure 4.9: Occurrences of the word "tekoäly" (artificial intelligence) in the STT corpus, years 1992–2018. Both overall word counts and the number of distinct articles containing the word are depicted.

number of distinct articles that mention the word, is quite low until it undergoes an exponential growth in the years leading up to 2018. The yearly word count for "tekoäly" between 1992 and 2013 is between 1 and 13, with an average of just over 5, hitting zero in the years 1994, 2010 and 2012.

A closer look of the last eight years of the word counts for "tekoäly" is seen in picture 4.10. I have also drawn these word counts again for the Yle corpus for the same years; only the word counts are depicted, not the article counts. We can see that the word counts for "tekoäly" are very similar for both corpora.

Based on the similarity of these word counts, we may reasonably hypothesise

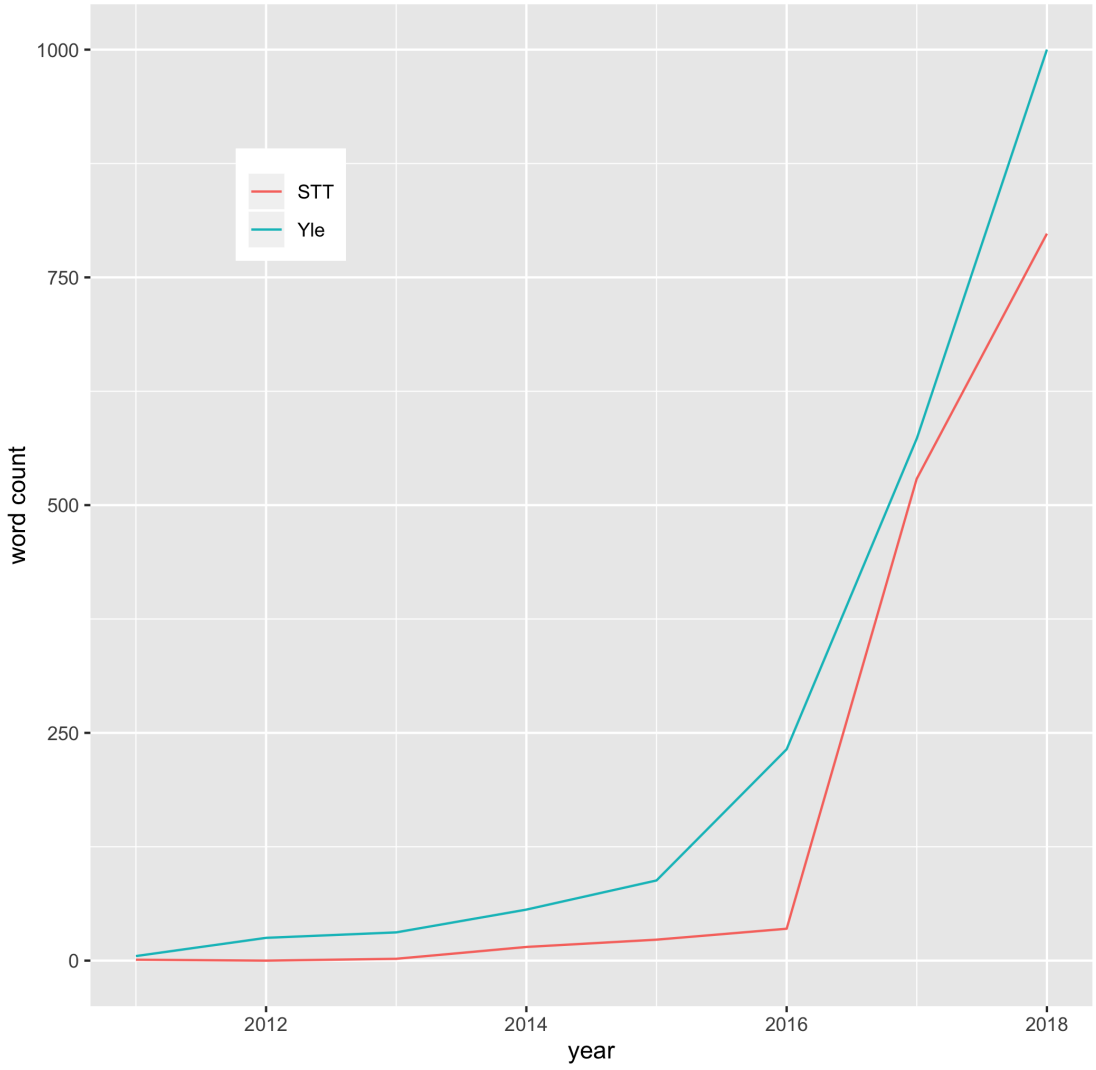

Figure 4.10: Occurrences of the word "tekoäly" (artificial intelligence) in both Yle and STT corpora, years 2011–2018 only.

that the behaviour of the word "tekoäly" in the STT corpus will be similar to its behaviour in the Yle corpus: in the years before 2014, it presumably refers to various, somewhat isolated practical applications, whereas by 2018, it can be predicted to have gained a more general status. Of course this is merely a guess, which we shall check in the following. The method of analysis is the same as before, i.e. that described in section 4.1: the analysis consists of a straightforward comparison of the nearest neighbours of the target word, across word2vec models which are trained on yearly batches of source material.

After training the yearly models from the STT corpus, I chose the years 1997, 2008 and 2018 as the years of interest. In this corpus, as we saw in figure 3.1, the

Neighbouring word	Distance	English description
tekoäly (1997)		
diffraktiivinen	0.6644	diffractive
membrane	0.6575	membrane
näköjärjestelmä	0.6553	vision system
lektiinihistokemiallinen	0.6518	lectin histochemical
pintafysiikka	0.6476	surface physics
neuraaliverkko	0.6473	neural network
duplex	0.6389	duplex
vesaluoma	0.6356	Vesaluoma (name)
kuvaanalyysi	0.6356	image analysis
epälineaarinen	0.6342	nonlinear
tekoäly (2008)		
automaatti–vaihteinen	0.6007	with automatic transmission
taltioiva	0.599	recording (present participle)
cagney	0.5984	Cagney (name)
heardi	0.5971	Heard (name)
goodrich	0.5947	Goodrich (company)
kappleri	0.5944	Kappler (name)
amerikanpitbullterrieri	0.5944	American pitbull terrier
vuoro-pohjainen	0.5891	turn-based
bondteema	0.5868	Bond [movie] theme
pelikirja	0.5861	games book
tekoäly (1997 words in 2008 model)		
diffraktiivinen	0.3164	diffractive
membrane	0.3255	membrane
näköjärjestelmä	0.3579	vision system
lektiinihistokemiallinen	0.4058	lectin histochemical
pintafysiikka	0.3018	surface physics
neuraaliverkko	0.3471	neural network
duplex	0.3518	duplex
vesaluoma	0.0854	Vesaluoma (name)
kuvaanalyysi	0.2934	image analysis
epälineaarinen	0.2119	nonlinear

Table 4.11: Nearest neighbours of "tekoäly" (artificial intelligence) in the STT corpus, years 1997 & 2008.

word and article counts are somewhat low prior to 1997, in which year they reach their "normal" level. The year 2018 is of course the last year for which material was available, and 2008 is (roughly) in between.

Table 4.11 lists, to begin with, the nearest neighbours of the word "tekoäly" in the STT corpus for the years 1997 and 2008, as well as the distance of each

1997 word in the 2008 model. Some artifacts caused by lemmatisation again occur, such as extraneous or missing dashes. Note that some of the neighbouring words are in English in the source material, rather than Finnish; this is because STT publishes news bulletins from various universities, which include titles of talks, doctoral theses and scientific papers in English.

From table 4.11 we see that the context of the word "tekoäly" remains fairly similar between 1997 and 2008 in the STT material. In both cases, the nearest context words refer to fairly specific, niche concepts: scientific terms, names of various researches, and popular culture. Only a minority of the nearest neighbours are more generic terms. An interesting case out of these is the word "neuraaliverkko" (neural network). This is a very generic term, and we see that it is not retained in the nearest neighbours of "tekoäly" between 1997 and 2008. This is an example of how "tekoäly" back in 2008 had not yet come to mean anything very specific or common, as seen from its context, i.e. from its nearest neighbours. We can hypothesise that had "tekoäly" attained a more general, common status, then it would have more general terms as its neighbours. This is in fact what we found with the Yle material earlier, as seen in tables 4.2 and 4.3.

As before, the table also shows the distances in 2008 of the nearest neighbours of 1997, in the last segment. We see that by 2008 the movement away from the previously nearest words was very significant across the board. This means that the context of "tekoäly" was by no means solid in 1997.

Table 4.12 depicts the nearest neighbours of "tekoäly" in 2018, as well as the distances of the words nearest in 1997 according to the 2018 model. We can see that this time there is a significant change towards more general terms, as opposed to specific or niche concepts. More than half of the ten nearest neighbours now refer to general concepts – digitalisation, machine learning and so on – compared to just one or two for both 1997 and 2008. This is very similar to what we observed earlier in the Yle corpus. In the Yle case all of the 10 nearest neighbours of "tekoäly" in 2018 were general terms, while here for STT only a majority of them are; but the trend towards more general terms over time is the same.

In the last segment of table 4.12 we see how the words which were nearest in 1997 fare in the 2018 model. Interestingly, every one of these 1997 words is closer to "tekoäly" in the 2018 model than they were close to it in the 2008 model. Perhaps the greater closeness could be a sign that the 2018 model has stabilised to a certain region, more so than the 2008 one.

Finally, a comparison with the 2018 Yle results from earlier, which were depicted in table 4.3, shows that a few of the nearest neighbours of "tekoäly" in 2018 are the same in both Yle and STT corpora, namely "koneoppiminen" (machine learning) and "teknologia" (technology). This is another similarity between the two sets of results obtained from the two corpora.

Based on all this we can argue that, similar to what we saw for the Yle corpus before, the context of "tekoäly" has changed over time also for the STT corpus. Again, not only have the exact words changed, but also their category, from more specific words earlier on to more general ones in 2018. We can see that this change

Neighbouring word	Distance	English description
tekoäly (2018)		
teknologia	0.7802	technology
digitalisaatio	0.7677	digitalisation
nfc-etätunnistusteknologia	0.7648	NFC remote ID technology
mydogdna-testauspalvelu	0.7401	MyDogDNA testing service
digitaalinen	0.7352	digital
pilvipalvelu	0.7245	cloud service
koneoppiminen	0.7231	machine learning
aivotieto	0.7212	brain data
kasvuseulatyökalu	0.7166	Kasvuseula.fi tool
materiaalikehitys	0.7131	research & development of materials
tekoäly (1997 words in 2018 model)		
diffraktiivinen	0.5149	diffractive
membrane	0.3878	membrane
näköjärjestelmä	0.5571	vision system
lektiinihistokemiallinen	0.5318	lectin histochemical
pintafysiikka	0.4634	surface physics
neuraaliverkko	0.5763	neural network
duplex	0.5657	duplex
vesaluoma	0.1336	Vesaluoma (name)
kuvaanalyysi	0.4382	image analysis
epälineaarinen	0.4707	nonlinear

Table 4.12: Nearest neighbours of "tekoäly" (artificial intelligence) in the STT corpus, years 1997 & 2018.

is not an isolated incident specific to the Yle corpus, but in fact occurs more broadly.

I also computed a visualisation of the changes in the nearest neighbours, which is seen in picture 4.13. As before, I chose to only include some of the nearest neighbours for clarity. The red dots depict words that were among the $k = 10$ nearest neighbours of "tekoäly" in 1997, 2008 or 2018, and the blue arrow shows the location of the target word "tekoäly" itself over time. This picture was produced with UMAP as the dimensionality reduction technique and with neither PCA nor Procrustes alignment needed for preprocessing.

As before, I computed the visualisation with UMAP several times with varying parameters, and selected one of these to display. Interestingly, all of these visualisations ended up clustering the 1997 and 2018 vectors closer to each other than to the 2008 vectors. This mirrors the earlier observation that the 1997 words were in fact closer to "tekoäly" in the 2018 model than they were in the 2008 model. It can be speculated therefore that the 2008 model was not yet very stable. It has to be noted that the simplest explanation for the 1997 model being seemingly more

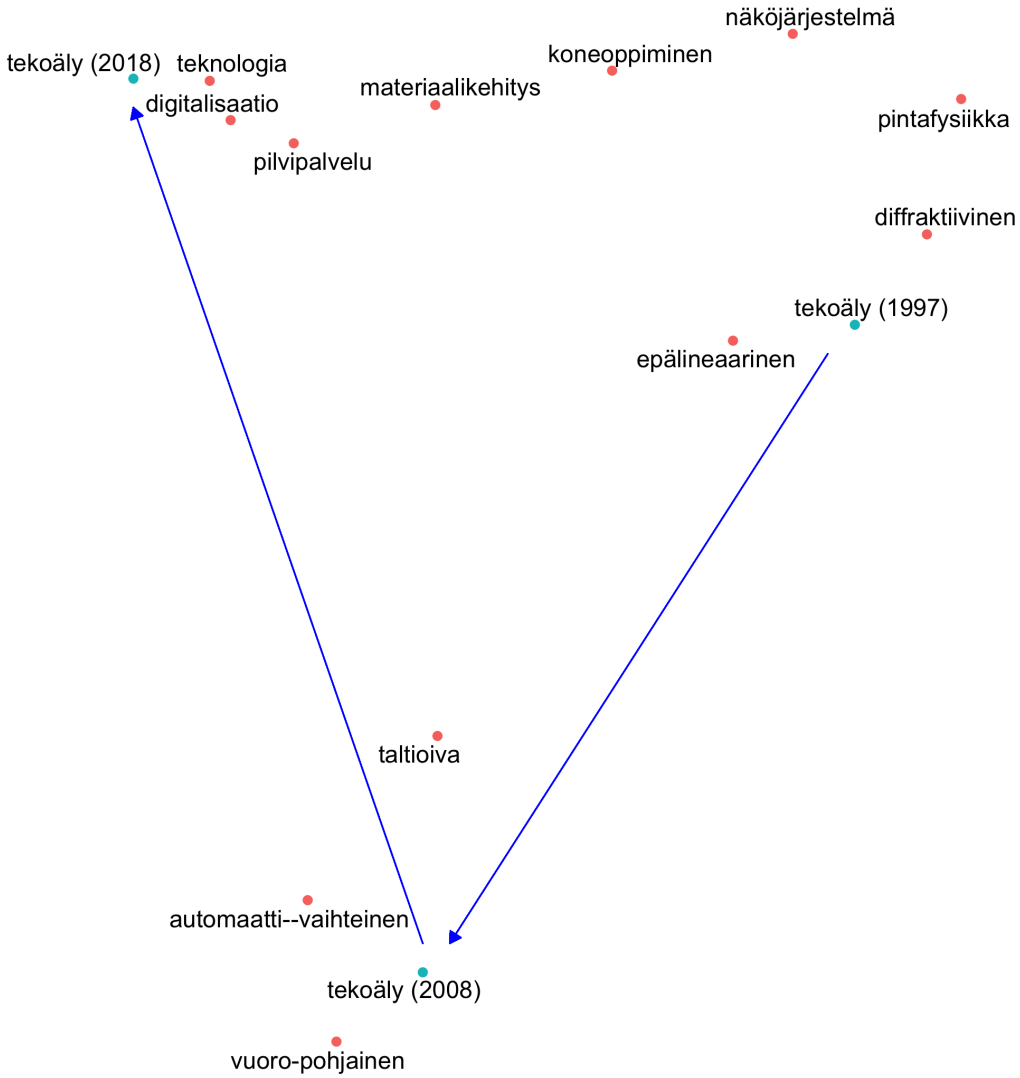

Figure 4.13: A comparison of the nearest neighbours of the word "tekoäly" (artificial intelligence) between 1997 and 2018, STT corpus. The red points indicate those words that were among "tekoäly"'s nearest neighbours in 1997, 2008 or 2018.

"stable" in this sense than the 2008 one is simply random chance, rather than any real effect; the low word occurrence counts for both 1997 and 2008 (see figure 4.9) would cast doubt upon the latter assumption.

4.3.1 Other interesting words

I continued the analysis with a look at one of the words that was among the $k = 10$ nearest neighbours of "tekoäly" in 2018, namely "koneoppiminen" (machine learning); in the STT results this 2018 neighbour was the closest one to

Neighbouring word	Distance	English description
koneoppiminen (1997)		
totuuskaltaisuus	0.7425	verisimilitude
optimointimalli	0.7416	optimisation model
skyrmi	0.7413	Skyrme (name)
häiriöteoria	0.738	perturbation theory
retentioindeksistandardi	0.7377	retention index standard
lääkeaineseulonta	0.736	drug screening
merkkijonotietokanta	0.7329	sequence database
laboratoriotesti	0.7275	laboratory test
keskustlullinen	0.724	conversational
ionisuihku	0.7239	ion shower
koneoppiminen (2018)		
koneoppiminenmenetelmä	0.7349	machine learning method
hiilipinta	0.7335	carbon surface
kiihtyvyysanturi-pohjainen	0.7262	accelerometer-based
tekoäly	0.7231	artificial intelligence
mallintaminen	0.7134	(technical) modelling
aivotieto	0.7102	brain data
viher-hoito	0.6953	green treatment
genomilaajuinen	0.6936	genome-wide
reunalaskenta	0.6923	edge computing
markkinointiteknologia	0.687	marketing technology

Table 4.14: Nearest neighbours of "koneoppiminen" (machine learning) in the STT corpus, years 1997 & 2018.

those in the Yle results earlier (see section 4.2). To begin with, a comparison of the $k = 10$ nearest neighbours of "koneoppiminen" in 1997 and in 2018 can be found in table 4.14. Notice that as ever there are some minor errors introduced by lemmatisation: the first word in "Skyrmen malli" (Skyrme's model, a physical theory) has become "skyrmi", and "keskustelullinen" (conversational) has become the erroneous "keskustlullinen". Again, these minor discrepancies do not affect the results.

We see that in 1997 the word "koneoppiminen" was associated with fairly specific, niche technical terms. In 2018 this was mostly still case, but in addition we have a few more general terms among the nearest neighbours, most significantly "tekoäly", but also "koneoppiminenmenetelmä" (machine learning method) and arguably "mallintaminen" (technical modelling). This result is very similar to what we saw for the term "tekoäly" itself earlier: in 1997, "koneoppiminen" was, judging by the context, a fairly niche technical concept, but by 2018 it had become somewhat more generic.

In figure 4.15 we can also see a visualisation of some of the nearest neighbours

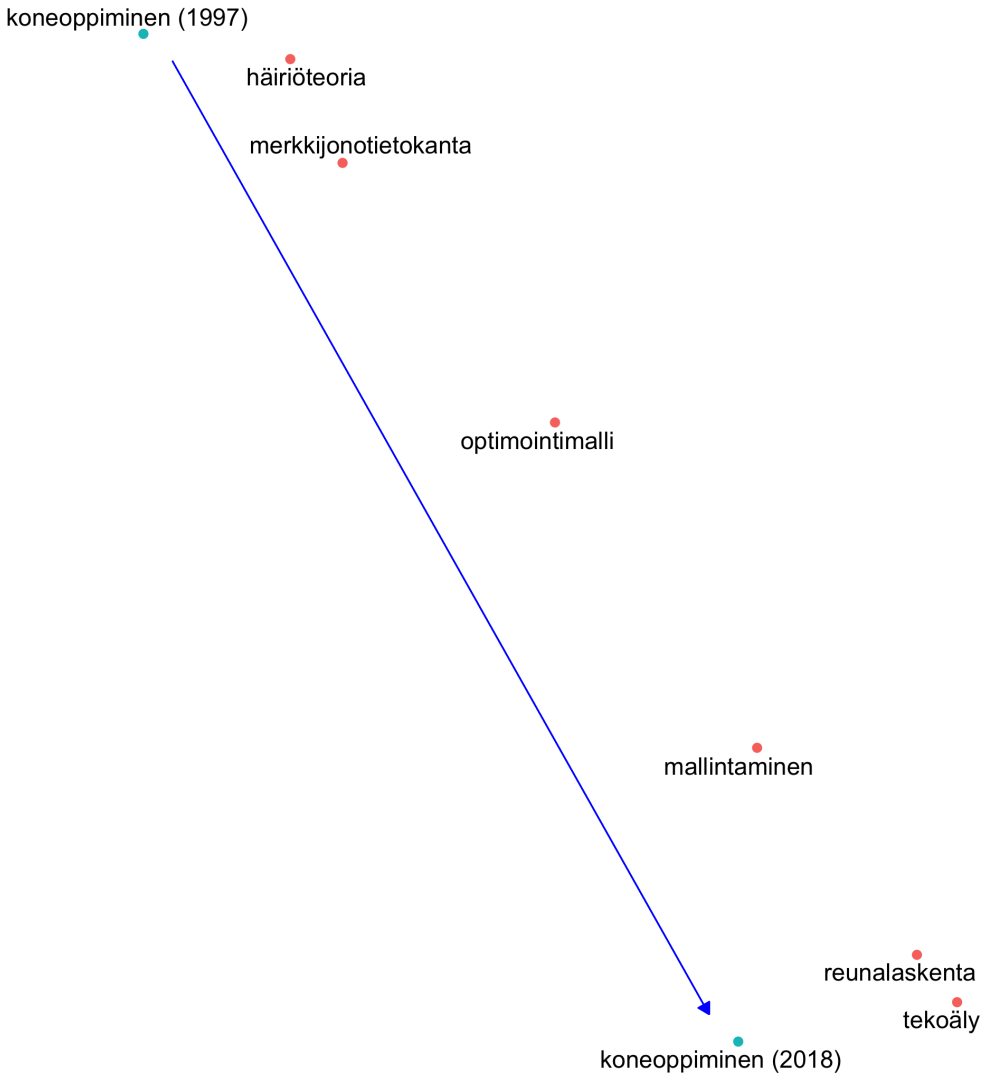

Figure 4.15: A comparison of the nearest neighbours of the word "koneoppiminen" (machine learning) between 1997 and 2018, STT corpus. The red points indicate those words that were among "koneoppiminen"'s nearest neighbours in 1997 or 2018.

of "koneoppiminen" in 1997 and 2018. For clarity, only some of the $k = 10$ nearest neighbours for each of the two years are depicted. This picture was produced with UMAP for dimensionality reduction, and neither PCA nor Procrustes alignment was required for preprocessing.

4.4 Stability and repeatability of results

One potential issue I wanted to investigate is the repeatability of the results when using word2vec. As was detailed in chapter 2, word2vec is a neural network, and as such it is inherently stochastic. Each time one trains a word2vec model, the starting weights, i.e. the word embeddings the model starts out with, are initialised randomly; this is necessary when using neural networks, as elaborated in standard textbooks on the matter (e.g. [27]). In the case of word2vec, various optimisations which were described in chapter 2 are also stochastic in nature, starting with the most important optimisation, negative sampling. Subsampling of frequent words is another optimisation that is probabilistic in nature. Finally, in skip-gram mode the algorithm can be set to use a context window of random length $R \sim \text{Unif}(1, C)$ rather than C, which is yet another source of randomness.

Because of all the randomness, the final word embeddings will also exhibit some variance across different model training sessions, even with the same source material and the same hyperparameters. While one could use the same random seed for each initialisation, this would not provide any guarantees as to the results with any other seed, and thus one could not tell whether these particular results are an outlier or a common case. Instead, the main feasible way to assess the repeatability of results is to run the analysis multiple times and gauge how much variation there is in different runs with different random seeds.

The main downside of repeating the analysis multiple times is the time and labour required. While in my case training the word2vec models from the two corpora I used is relatively quick, the time required obviously depends on the size of the source material and the computational power available; there may also be monetary costs to be considered with respect to the computation. As well, the analysis of the results can be laborious to repeat.

As detailed in chapter 2, several authors have investigated the convergence properties of word2vec, and some have found that word2vec with negative sampling is guaranteed to converge under certain conditions [52]. However, even if convergence is assumed, it is not clear how quickly it will happen.

4.4.1 Repeated analysis: Yle corpus

It turns out that in my case, for the two corpora I used, the results are in fact quite stable. I ran the previously detailed analysis again with the exact same source material and hyperparameters, so that the only thing that varied was the randomness present in the word2vec implementation itself. I then examined the nearest neighbours of the main target word, "tekoäly" (artificial intelligence), in the same way as before.

For the Yle corpus, the results of the rerun are depicted in table 4.16. Starting from the latest year, 2018, the results are remarkably similar to those from the first run, which were presented in table 4.3. In fact, out of the ten nearest neighbours of "tekoäly" in 2018, nine are the same across the two runs, while in the second run,

Neighbouring word	Distance	English description
tekoäly (2011)		
tietovisailu	0.6612	quiz; game show
moottoritekniikka	0.5875	motor technology
alkoholipolttoaine	0.5823	alcohol-based fuel
kerroshampurilainen	0.5821	double hamburger
maalaustaide	0.5819	painting [noun]
vähä-päästö	0.5813	low-emission
afrikkalaisuus	0.58	Africanness
ekojalanjälki	0.5791	ecological footprint
innovaatiopolitiikka	0.5788	innovation politics
soitinrakentaminen	0.5758	musical instrument making
tekoäly (2014)		
mtrn	0.6824	MTR (Hong Kong transport company)
algoritmi	0.6302	algorithm
turinki	0.622	Turing (last name)
icf	0.6137	Intelligent Community Forum
botti	0.6127	bot (i.e. "software robot")
amatöörimainen	0.6012	amateurish
tietokoneohjelma	0.5882	computer program
gmes	0.5754	GMES/Copernicus (ESA programme)
ohjelmoida	0.5748	to program
internetjätti	0.5747	Internet giant
tekoäly (2018)		
teknologia	0.8091	technology
syväoppiminen	0.7965	deep learning
koneoppiminen	0.7721	machine learning
algoritmi	0.7553	algorithm
koneoppiminenmenetelmä	0.7457	machine learning method
teknologinen	0.7341	technological
koneäly	0.7234	machine intelligence
robotti	0.7116	robot
super-tekoäly	0.7055	super AI (artificial intelligence)
keino-äly	0.7044	constructed intelligence

Table 4.16: Second run: nearest neighbours of "tekoäly" (artificial intelligence) in the Yle corpus, years 2011, 2014 & 2018.

the word "digitaaliteknologia" (digital technology) is missing and has been replaced in the top ten by "koneäly" (machine intelligence). The nearest neighbours are in different order and the distance of each neighbour from the target word "tekoäly" varies somewhat in the second run, but interestingly this variance is fairly small: in the first run, the minimum and maximum distances of the ten neighbours were

were 0.7048 and 0.7916 respectively, while for the second run they are 0.7044 and 0.8091, for a maximum difference of roughly 2 %. The content of the second run results, for 2018, can therefore to be said to be basically identical to those of the first run.

Looking at the results for 2014, we see that six out of ten nearest neighbours of "tekoäly" are preserved in the second run, when compared to the first run, which was documented in table 4.2. Again, the six shared words are in a different order and with slightly different distances, compared to the first run. The maximum difference between the minimum and maximum distances of the first run and the second run is again roughly 2 %. Considering the four new words which appear in the top ten of 2014 in the second run, three of them – "turinki" (a wrongly lemmatised Turing, as in Alan Turing), "amatöörimainen" (amateurish), and "gmes" (GMES, the former name of Copernicus, an European Space Agency earth monitoring programme) – are fairly specific concepts, while "internetjätti" (Internet giant) is somewhat more general. As all of the more general words from the first run are still present, it can be said that the balance between specific concepts and more general terms in the 2014 Yle results has not changed on the second run.

For 2011, the results of the rerun can be seen to differ significantly from the previous run's results. While the difference in the minimum and maximum distances is less than 2 % this time, out of the ten nearest neighbours, nine have changed. Only "tietovisailu" (quiz; game show) is in the top ten for both runs; interestingly, it is also in the top spot for both. In any case, the larger changes here, compared to the other two years, is consistent with the fact that the target word, "tekoäly", is simply not very common yet in the year 2011 source material, and the results for that word are not very stable. As we saw, the results become more stable for 2014 and much more stable for 2018. While the exact words differ greatly for 2011 across the two runs, the word categories remain the same: in both cases, "tekoäly" co-occurred almost exclusively with words referring to specific, relatively niche applications, rather than broader terms and concepts.

4.4.2 Repeated analysis: STT corpus

The results of the rerun on the STT corpus are shown in table 4.17. For 2018, the first run's results for STT were depicted in table 4.12, and when comparing, we again see a large overlap between the two sets of results. Seven out of ten words for 2018 are the same in the second run, with a different order and a fairly small difference in the distances of less than 2 %. Interestingly, two of the three words that have dropped out of the top ten are fairly specific concepts – "aivotieto" (brain data) and "materiaalikehitys" (research & development of materials) – and two of the three new words that appear in the rerun are also fairly specific: "käyttötapaus" (use case) and "intensiivikoulutus" (intensive training). The remaining word that fell out in the second run, "pilvipalvelu" (cloud service), is more general, and so is the remaining new word, "digitalisoitua" (to become digitalised). Therefore the rerun results retain the same division into more general and more specific

Neighbouring word	Distance	English description
tekoäly (1997)		
epälineaarinen	0.6637	nonlinear
neuraaliverkko	0.6441	neural network
lektiinihistokemiallinen	0.6428	lectin histochemical
joutsensalo	0.6339	Joutsensalo (name)
bioreseptori	0.6305	bioreceptor
näköjärjestelmä	0.6261	vision system
hepatic	0.6252	hepatic
jia-qing	0.624	Jia-Qing (name)
molekylaarinen	0.6229	molecular
koneoppiminen	0.6227	machine learning
tekoäly (2008)		
karkotin	0.6332	repellent
malakian	0.6326	Malakian (name)
saudinainen	0.6168	Saudi woman
torjuntatyyli	0.6039	blocking style
artico	0.5995	Artico (name)
syöttäjätähti	0.5908	star pitcher
jalkapallohenkilö	0.5889	football personality
magneettiaisti	0.5884	magnet instinct
kolmemiehinen	0.5883	three-man
rocksävelmä	0.5881	rock tune
tekoäly (2018)		
nfc-etätunnistusteknologia	0.7722	NFC remote ID technology
teknologia	0.7714	technology
digitalisaatio	0.7589	digitalisation
koneoppiminen	0.7383	machine learning
digitaalinen	0.7201	digital
digitalisoitua	0.7144	to become digitalised
mydogdna-testauspalvelu	0.7135	MyDogDNA testing service
käyttötapaus	0.7081	use case
intensiivikoulutus	0.7064	intensive training
kasvuseulatyökalu	0.7041	Kasvuseula.fi tool

Table 4.17: Second run: nearest neighbours of "tekoäly" (artificial intelligence) in the STT corpus, years 1997, 2008 & 2018.

concepts as the first run did, where the majority of the nearest neighbours in 2018 are general terms, as opposed to specific concepts. Perhaps the greater stability of the more general words across reruns can be seen to strengthen the result that by 2018, the word "tekoäly" referred to more general concepts, as opposed to more specific ones as it did earlier.

For 2008, the STT corpus gives very different results on the rerun. The previous run's results for 2008 were depicted in table 4.12; on the rerun, all ten words are different and the distances differ by more than 5 %. Examining the 2008 words of the rerun, we see that all of them refer to specific, niche concepts, which is what happened also on the first run. In the original analysis, we saw that for STT the year 2008 was an outlier in terms of the closest neighbours: the words which were closest to "tekoäly" in 1997 were closer to "tekoäly" in 2018 than were the closest neighbours from 2008. This second analysis seems to corroborate this finding: in 2008, the nearest neighbours are definitely not stable.

Examining the 1997 words, we see greater stability compared to 2008, but not that much stability overall. On the second run of STT 1997, four of the ten words have remained the same as in the first run. The maximum difference in the distances is less than 2 % between the runs, which is small compared to the previous rerun results we have discussed. Of the six new words, five are specific, niche terms, but the remaining one, "koneoppiminen" (machine learning) is clearly more generic. This word however is not enough to change the year's results: these 1997 terms remain scattered and specific in the second run, with only a minority being more general terms. In this sense, the second run's results match those of the first run.

In all, the rerun indicates that the results obtained from the analysis detailed in this chapter are genuine, in the sense that they are not due to any random variation in how word2vec works as an algorithm. Although there are differences, sometimes large differences, in the results between the two runs, the conclusions remain the same. Indeed, the differences we have noted seem to indicate which parts of our results are stable and which are less so. This kind of repeated analysis could offer a valuable way of gaining further confidence in the results of experiments done with word2vec.

4.5 Summary

We arrive at the conclusion that the usage of the word "tekoäly", as determined by its context, has noticeably changed by the year 2018, whether one begins the examination from 1997, 2008 or 2011. This change in usage has occurred in both the Yle and the STT corpus, and it remains largely the same in two different repeat analyses. We can therefore argue further that this observed change in context, and therefore the meaning, of the word "tekoäly" is neither a case of a single media company using the word differently, nor due to random variation in the word2vec algorithm, but rather a broader, genuine change.

It is interesting that the word2vec method is able to capture not only this general trend in both corpora examined, but also the differences between the corpora. Comparing tables 4.3 and 4.12 we can see that the overlap between the nearest neighbours between Yle and STT is quite small, if one goes by the exact specific words themselves. However, if we consider which types of word these neighbours

are, we see a marked commonality between the corpora, yet with the difference that the STT corpus seems less "solidified" regarding the place of "tekoäly" in it: in the STT corpus, "tekoäly" has not, perhaps not yet, ended up as a fully generic, maximally broad concept, whereas this seems to have occurred, or in any case to a noticeably greater extent, in the Yle corpus.

In Hamilton et al.'s paper [32], which worked as an inspiration for this chapter, the findings are of course more robust. This is due to the much larger corpora used in their paper, both in overall volume and the number of years studied. It is interesting to note that such analysis can be undertaken even with smaller corpora – the results presented above were fairly clear, despite the much smaller word counts across many fewer years. This suggests that word2vec is a reasonably robust method of analysis for context change even on a smaller scale. We also see that word2vec can handle the Finnish language fine, at least when the material is lemmatised first.

Chapter 5

Discussion

As was discussed in chapter 2, what word2vec computes is an approximation of the distribution of context words, given an input of sentences. While calculating this distribution is not in itself a complicated feat, making sense of it is where the real "added value" lies. The word vectors provided by word2vec are constructed so as to provide a distance metric between any two words, based on the similarity (or otherwise) of their context. This distance metric is an easy, practical way to estimate the similarity of any given words in a text corpus. One can also, as we did in chapter 4, examine how this similarity changes over time. Thus, word2vec is a usable method for examining, not the context distribution directly, but rather its impact.

In addition to this, it turns out that the word embeddings computed by word2vec are useful for many other tasks, as was detailed in 2.5 . This is because the embeddings contain a lot of information about the word contexts of the source material. This informativeness makes them a very widely applicable, versatile tool.

5.1 Word2vec and other algorithms

Word2vec has proven to be a powerful, yet relatively straightforward algorithm. It has been used for many different applications and has in turn inspired a great number of improved versions, hybrid "word2vec-plus-something" algorithms and novel feature learning algorithms.

In chapter 1 we discussed word2vec in the context of other models and algorithms used for natural language processing (NLP). We saw that word2vec is often used in an indirect way, which puts it in a somewhat different category than, say, topic models, which were designed for a specific task. Of course, the word embeddings produced by word2vec can also be examined directly, as was done in chapters 3 and 4.

Word2vec is by no means the only algorithm to take advantage of the context distribution; GloVe [66] is another notable example. Word embeddings are a very

active topic of reseach due to their wide applicability. Whether proceeding from the context distribution, as GloVe does, or from a different direction such as word2vec or its successors (e.g. [59]), the theoretical foundation of word embeddings in general is currently being strengthened in various ways. In addition, the various uses and applications of word embeddings are under heavy investigation as well.

While word embeddings are not the only NLP tool, due to their versatility they are one of the biggest, being used in many NLP tasks from examinations of evolving semantics (chapter 4) to more direct investigations of context [24, 77].

5.2 Word2vec in practice

One reason for word2vec's popularity is undoubtedly its ease of use for practical tasks, which I have verified for myself. In chapter 3 we saw how word2vec can be used as part of a practical language processing pipeline. As natural language processing algorithms go, word2vec is very straightforward to use. This is thanks in no small part to the well-working gensim implementation [68], but also due to the general simplicity of word2vec in general. Word2vec does not require extensive preprocessing, beyond the standard preprocessing which most NLP methods generally require such as lowercasing, removal of punctuation, possibly lemmatising etc. Training the models does not take onerously long, at least using the gensim implementation. I did not run into any significant issues in the practical usage of word2vec with the gensim implementation.

It is also relatively straightforward to write a custom word2vec implementation, which obviously makes research and experimentation easier. Indeed, some machine learning programming frameworks, such as TensorFlow [1], provide a tutorial on how to implement word2vec in the framework in question [75], as implementing word2vec will work as a reasonable introduction to learning how to accomplish useful tasks with the framework. Thus word2vec these days could be called a standard algorithm.

Due to its relative ease of use and effectiveness, as seen in my experiences which are documented in this thesis, I can recommend word2vec as an NLP tool for tasks which can benefit from examining the context distribution, such as an examination of the changing of the context distribution over time, which I documented in chapter 4. In general, word2vec can work well for any NLP task that operates on the sentence level of text or is related to the co-occurrences and contexts of words.

Many applications can also benefit from word embeddings in a more indirect way, as we saw in section 2.5. In such cases one should consider whether precomputed word embeddings could be used, as this will significantly lower the barrier to entry. The downside of precomputed embeddings is that they of course cannot be used to analyse custom material, i.e. material that they were not trained with. Because of this, precomputed embeddings answer a different question than that of a custom text analysis – they are useful for enriching fairly "standard", ordinary text for further processing, rather than directly analysing said text itself.

5.3 Experiment: semantic change over time

In chapter 4 an experiment was presented where word2vec was used to attempt to gauge the change in meaning of a certain concept over time. To accomplish this a sequence of word2vec models was trained, one for each year of the source material, to investigate the changes of word contexts over time, for words related to the target concept.

We concluded that the usage of the word "tekoäly" (Finnish for "artificial intelligence") had changed noticeably over time, as examined via word2vec models trained on two different corpora of Finnish news articles. Word2vec was able to detect this change in context and readily provided concrete examples of the changes for each year of material. Furthermore, we were able to differentiate the two corpora with respect to the level of change they showed for the target word. In a repeat analysis, the results remained the same, meaning that word2vec performed in a stable way on this particular problem. Despite the relatively small size of each yearly batch of material, word2vec performed well and produced useful, understandable results.

The expreriment described in chapter 4 was successful enough that, in my opinion, word2vec can be recommended as a quantitative method for other researchers as well: it is both easy to use and powerful. Indeed, looking at the bigger picture, some researchers have attempted to use such quantitative NLP techniques together with, and as a basis for, qualitative analysis [63]. This is the general approach taken by researchers such as Ylä-Anttila [83], who uses automated computational methods of text analysis as a building block of a larger analytical system. While Ylä-Anttila utilises topic modelling, others have instead used word2vec as a building block for hybrid systems that incorporate qualitative analysis into a system built on word2vec as well as novel algorithms [28, 44]. In general, word2vec is robust enough to qualify as a basic building block for such combined methods.

5.4 Limitations of word2vec

While word2vec is a useful and versatile algorithm, it has its limitations. These limitations were discussed in section 2.4 and later developments to develop more advanced algorithms were discussed in section 2.6.

The main drawback of word2vec is that it assigns each unique token in the source material precisely one embedding vector, regardless of possible differences in word sense (polysemy), part of speech or inflection. This makes word2vec unsuitable for some specialised use cases such as natural language generation. Fortunately, for such more advanced problems, newer algorithms exist, such as fastText [6, 40] and BERT [14] to name just a few, as was discussed in section 2.6.

In section 4.4 we discussed another potential caveat of word2vec, namely the repeatability of its results. Word2vec is a stochastic algorithm and its results are therefore somewhat unpredictable. While the embeddings produced by word2vec

do eventually converge to the context distribution, it is not clear a priori how repeatable this convergence is on repeat runs of the algorithm on the same material, or how rapidly the algorithm converges. However, in practice this is not a problem, as was demonstrated in chapter 4: word2vec was in this case found to converge both at a practical speed (i.e. fast enough, without needing excessively many training epochs) and repeatably. These experiences mirror those of other practitioners who have used word2vec. It can therefore be said, based on my experiences as documented in this thesis, that word2vec is a stable and robust algorithm.

Chapter 6

Conclusion

Word2vec is an algorithm for finding compressed representations of word contexts from natural-language text. As we saw, these compressed representations, word embeddings, have many applications and uses. The one we focused on in this thesis was a straightforward examination of the word embeddings to study the corresponding changes in word context over time.

Our hypothesis in chapter 4 was that if the meaning of a certain concept has changed over time, then this change will be reflected in corresponding changes in word contexts, for those words that are most closely related to that concept. While the word contexts are undoubtedly an imperfect proxy for the true meaning of the concept, however that is defined, they can nevertheless be a useful approximation. Much of this usefulness comes from the fact that the word contexts can be measured and analysed, and this can be done even for very large masses of source material. Word2vec is a suitable tool for such analysis.

In chapter 4 our conclusion was that the usage of our target word, "tekoäly" (artificial intelligence), has indeed changed over time in both source corpora. We were able to quantify this change, in a repeatable and robust manner, and could therefore argue further that this represents a true change in the meaning of the term "tekoäly". Word2vec proved to be very capable of detecting and measuring such changes, being able to also differentiate the corpora from one another. A repeat analysis also allowed us to judge the stability of the word2vec results for each year studied. All in all, word2vec was proven to perform quite well on such context change tasks even with relatively small amounts of source material.

While word2vec is not perfect, it provides, in my experience as documented in this thesis, very good "value for money", in the sense that it is both understandable and powerful. For this reason I can recommend obtaining at least basic familiarity with the ideas and techniques of word2vec, since they are useful in their own right and also shed light on the exact workings of later algorithms. Another point in word2vec's favour is the fact that, as a well understood algorithm, it is both straightforward to use in practice, thanks to the well-functioning gensim implementation, and also simple enough to build a custom implementation for should one be needed, or to learn more about this type of algorithm in general.

Bibliography

[1] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Kudlur, M. (2016). TensorFlow: A system for large-scale machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)* (pp. 265–283).

[2] Arora, S., Li, Y., Liang, Y., Ma, T., & Risteski, A. (2018). Linear algebraic structure of word senses, with applications to polysemy. *Transactions of the Association for Computational Linguistics, 6*, 483–495.

[3] Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research, 3* (Feb), 1137–1155.

[4] Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research, 3* (Jan), 993–1022.

[5] Blei, D. M. (2012). Probabilistic topic models. *Communications of the ACM, 55* (4), 77–84.

[6] Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics, 5*, 135–146.

[7] Bolukbasi, T., Chang, K. W., Zou, J. Y., Saligrama, V., & Kalai, A. T. (2016). Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings. In *Advances in neural information processing systems* (pp. 4349–4357).

[8] Bray, T. (2017). RFC 8259: The JavaScript Object Notation (JSON) Data Interchange Format. Internet Engineering Task Force (IETF).

[9] Caliskan, A., Bryson, J. J., & Narayanan, A. (2017). Semantics derived automatically from language corpora contain human-like biases. *Science, 356(6334)*, 183–186.

[10] Ching, T., Himmelstein, D. S., Beaulieu-Jones, B. K., Kalinin, A. A., Do, B. T., Way, G. P., ... & Xie, W. (2018). Opportunities and obstacles for deep

learning in biology and medicine. *Journal of The Royal Society Interface, 15(141)*, 20170387.

[11] Cordeiro, S., Ramisch, C., Idiart, M., & Villavicencio, A. (2016, August). Predicting the compositionality of nominal compounds: Giving word embeddings a hard time. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (pp. 1986–1997).

[12] Corral, A., Boleda, G., & Ferrer-i-Cancho, R. (2015). Zipf's law for word frequencies: Word forms versus lemmas in long texts. *PloS one, 10* (7).

[13] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., & Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American society for information science, 41* (6), 391–407.

[14] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

[15] Dietterich, T. (1995). Overfitting and undercomputing in machine learning. *ACM computing surveys (CSUR), 27(3)*, 326–327.

[16] DiMaggio, P. (2015). Adapting computational text analysis to social science (and vice versa). *Big Data & Society, 2* (2), 2053951715602908.

[17] Dubin, D. (2004). The most influential paper Gerard Salton never wrote.

[18] Dumais, S. T., Furnas, G. W., Landauer, T. K., Deerwester, S., & Harshman, R. (1988, May). Using latent semantic analysis to improve access to textual information. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 281–285). Acm.

[19] Dyer, C. (2014). Notes on noise contrastive estimation and negative sampling. *arXiv preprint arXiv:1410.8251*.

[20] Fast, E., & Horvitz, E. (2017, February). Long-term trends in the public perception of artificial intelligence. In *Thirty-First AAAI Conference on Artificial Intelligence*.

[21] Firth, J. R. (1957). A synopsis of linguistic theory, 1930–1955. *Studies in linguistic analysis*.

[22] Flood, B. J. (1999). Historical note: the start of a stop list at Biological Abstracts. *Journal of the Association for Information Science and Technology, 50(12)*, 1066.

[23] Garg, N., Schiebinger, L., Jurafsky, D., & Zou, J. (2018). Word embeddings quantify 100 years of gender and ethnic stereotypes. *Proceedings of the National Academy of Sciences, 115(16)*, E3635–E3644.

[24] Gligorijevic, D., Stojanovic, J., Djuric, N., Radosavljevic, V., Grbovic, M., Kulathinal, R. J., & Obradovic, Z. (2016). Large-scale discovery of disease-disease and disease-gene associations. *Scientific reports, 6(1)*, 1–12.

[25] Goldberg, Y., & Levy, O. (2014). word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*.

[26] Gonen, H., & Goldberg, Y. (2019). Lipstick on a pig: Debiasing methods cover up systematic gender biases in word embeddings but do not remove them. *arXiv preprint arXiv:1903.03862*.

[27] Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). *Deep learning* (Vol. 1, p. 2). Cambridge: MIT press.

[28] Gorro, K., Ancheta, J. R., Capao, K., Oco, N., Roxas, R. E., Sabellano, M. J., ... & Goldberg, K. (2017, December). Qualitative data analysis of disaster risk reduction suggestions assisted by topic modeling and word2vec. In *2017 International Conference on Asian Language Processing (IALP)* (pp. 293–297). IEEE.

[29] Grbovic, M., & Cheng, H. (2018, July). Real-time personalization using embeddings for search ranking at airbnb. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (pp. 311–320).

[30] Grimmer, J., & Stewart, B. M. (2013). Text as data: The promise and pitfalls of automatic content analysis methods for political texts. *Political analysis, 21* (3), 267–297.

[31] Gutmann, M. U., & Hyvärinen, A. (2012). Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *The journal of machine learning research, 13(1)*, 307–361.

[32] Hamilton, W. L., Leskovec, J., & Jurafsky, D. (2016). Diachronic word embeddings reveal statistical laws of semantic change. *arXiv preprint arXiv:1605.09096*.

[33] Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: data mining, inference, and prediction*, pp. 241–249. Springer Science & Business Media.

[34] Haverinen, K. *Natural Language Processing Resources for Finnish* (Doctoral dissertation, PhD thesis, University of Turku, 8 2014).

[35] Hinton, G. E., McClelland, J. L., & Rumelhart, D. E. (1984). *Distributed representations* (pp. 1–127). Pittsburgh, PA: Carnegie-Mellon University.

[36] Hofmann, T. (1999, August). Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval* (pp. 50–57).

[37] Hofmann, T. (2001). Unsupervised learning by probabilistic latent semantic analysis. *Machine learning, 42* (1-2), 177–196.

[38] Hyvärinen, I. (2019). Compounds and multi-word expressions in Finnish. *Complex lexical units: Compounds and multi-word expressions*, 307–335.

[39] Häkkinen, K. (1994). *Kielitieteen perusteet* (Vol. 133). Suomalaisen Kirjallisuuden Seura.

[40] Joulin, A., Grave, E., Bojanowski, P., & Mikolov, T. (2016). Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*.

[41] Jurafsky, D., & Martin, J. H. (2008). *Speech and Language Processing: An introduction to speech recognition, computational linguistics and natural language processing*. Upper Saddle River, NJ: Prentice Hall.

[42] Jurafsky, D., & Martin, J. H. (2020 [draft]). *Speech and Language Processing: An introduction to speech recognition, computational linguistics and natural language processing (3rd Edition)*. Retrieved from https://web.stanford.edu/ jurafsky/slp3/.

[43] Kearns, M., Mansour, Y., Ng, A. Y., & Ron, D. (1997). An experimental and theoretical comparison of model selection methods. *Machine Learning, 27(1)*, 7–50.

[44] Kim, S., Park, H., & Lee, J. (2020). Word2vec-based latent semantic analysis (W2V-LSA) for topic modeling: a study on blockchain technology trend analysis. *Expert Systems with Applications*, 113401.

[45] Kim, Y., Chiu, Y. I., Hanaki, K., Hegde, D., & Petrov, S. (2014). Temporal analysis of language through neural language models. *arXiv preprint arXiv:1405.3515*.

[46] Korenius, T., Laurikkala, J., Järvelin, K., & Juhola, M. (2004, November). Stemming and lemmatization in the clustering of finnish text documents. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management* (pp. 625–633).

[47] Kulkarni, V., Al-Rfou, R., Perozzi, B., & Skiena, S. (2015, May). Statistically significant detection of linguistic change. In *Proceedings of the 24th International Conference on World Wide Web* (pp. 625–635).

[48] Kutuzov, A., Øvrelid, L., Szymanski, T., & Velldal, E. (2018). Diachronic word embeddings and semantic shifts: a survey. *arXiv preprint arXiv:1806.03537*.

[49] Le, Q., & Mikolov, T. (2014, January). Distributed representations of sentences and documents. In *International conference on machine learning* (pp. 1188–1196).

[50] Leeuwenberg, A., Vela, M., Dehdari, J., & van Genabith, J. (2016). A minimally supervised approach for synonym extraction with word embeddings. *The Prague Bulletin of Mathematical Linguistics, 105* (1), 111–142.

[51] Levy, O., & Goldberg, Y. (2014). Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems* (pp. 2177–2185).

[52] Li, Y., Xu, L., Tian, F., Jiang, L., Zhong, X., & Chen, E. (2015, June). Word embedding revisited: A new representation learning and explicit matrix factorization perspective. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

[53] Maaten, L. V. D., & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of machine learning research, 9(Nov)*, 2579–2605.

[54] MacQueen, J. (1967, June). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability* (Vol. 1, No. 14, pp. 281–297).

[55] Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to information retrieval*. Cambridge university press.

[56] McInnes, L., Healy, J., & Melville, J. (2018). Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*.

[57] Mikolov, T. (2013, 7 Oct). Re: de-obfuscated Python + question [online discussion comment]. Message posted to https://groups.google.com/forum/?#!topic/word2vec-toolkit/NLvYXU99cAM.

[58] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

[59] Mikolov, T., Le, Q. V., & Sutskever, I. (2013). Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168*.

[60] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (pp. 3111–3119).

[61] Mitra, B., Nalisnick, E., Craswell, N., & Caruana, R. (2016). A dual embedding space model for document ranking. *arXiv preprint arXiv:1602.01137*.

[62] Mäkelä, E. (2016). LAS: an integrated language analysis tool for multiple languages. *J. Open Source Software, 1* (6), 35.

[63] Muller, M., Guha, S., Baumer, E. P., Mimno, D., & Shami, N. S. (2016, November). Machine learning and grounded theory method: Convergence, divergence, and combination. In *Proceedings of the 19th International Conference on Supporting Group Work* (pp. 3–8).

[64] Nalisnick, E., Mitra, B., Craswell, N., & Caruana, R. (2016, April). Improving document ranking with dual word embeddings. In *Proceedings of the 25th International Conference Companion on World Wide Web* (pp. 83–84).

[65] Nissim, M., van Noord, R., & van der Goot, R. (2019). Fair is Better than Sensational: Man is to Doctor as Woman is to Doctor. *arXiv preprint arXiv:1905.09866*.

[66] Pennington, J., Socher, R., & Manning, C. (2014, October). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532–1543).

[67] Pentzold, C., Brantner, C., & Fölsche, L. (2019). Imagining big data: Illustrations of "big data" in US news articles, 2010–2016. *New Media & Society, 21* (1), 139–167.

[68] Rehurek, R., & Sojka, P. (2010). Software framework for topic modelling with large corpora. In *In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*.

[69] Rong, X. (2014). word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*.

[70] Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics, 20*, 53–65.

[71] Salton, G. (Ed.). (1971). *The SMART retrieval system: Experiments in automatic document processing*. Englewood Cliffs, NJ: Prentice-Hall.

[72] Schönemann, P. H. (1966). A generalized solution of the orthogonal procrustes problem. *Psychometrika, 31(1)*, 1–10.

[73] STT. *Finnish News Agency Archive 1992-2018, source* [text corpus]. Kielipankki. Retrieved from http://urn.fi/urn:nbn:fi:lb-2019041501

[74] Sutskever, I., Martens, J., & Hinton, G. E. (2011, January). Generating text with recurrent neural networks. In *ICML*.

[75] TensorFlow (2020, 24 Sep). Word embeddings | TensorFlow Core. *TensorFlow*. Retrieved from `https://www.tensorflow.org/tutorials/text/word_embeddings`.

[76] Trask, A., Michalak, P., & Liu, J. (2015). sense2vec – a fast and accurate method for word sense disambiguation in neural word embeddings. *arXiv preprint arXiv:1511.06388*.

[77] Tshitoyan, V., Dagdelen, J., Weston, L., Dunn, A., Rong, Z., Kononova, O., ... & Jain, A. (2019). Unsupervised word embeddings capture latent knowledge from materials science literature. *Nature, 571* (7763), 95.

[78] Wang, J., Huang, P., Zhao, H., Zhang, Z., Zhao, B., & Lee, D. L. (2018, July). Billion-scale commodity embedding for e-commerce recommendation in alibaba. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (pp. 839–848).

[79] Wilkerson, J., & Casas, A. (2017). Large-scale computerized text analysis in political science: Opportunities and challenges. *Annual Review of Political Science, 20*, 529–544.

[80] Wu, L., Fisch, A., Chopra, S., Adams, K., Bordes, A., & Weston, J. (2017). Starspace: Embed all the things!. *arXiv preprint arXiv:1709.03856*.

[81] Yao, Z., Sun, Y., Ding, W., Rao, N., & Xiong, H. (2018, February). Dynamic word embeddings for evolving semantic discovery. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining* (pp. 673–681).

[82] Yleisradio. *Yle Finnish News Archive 2011-2018, source* [text corpus]. Kielipankki. Retrieved from http://urn.fi/urn:nbn:fi:lb-2017070501

[83] Ylä-Anttila, T. S. S. (2017). The Populist Toolkit: Finnish Populism in Action 2007–2016.

[84] Zhang, Y., Chen, Q., Yang, Z., Lin, H., & Lu, Z. (2019). BioWordVec, improving biomedical word embeddings with subword information and MeSH. *Scientific data, 6(1)*, 1–9.

Appendix A

Finnish analogy set

This appendix details the set of Finnish-language analogies used to evaluate word2vec models. The process was described in detail in section 3.2.1.

The following is a segment of Python code documenting both the analogy pairs used and the method with which these pairs were turned into tuples of four words each for use in evaluation.

```
# generate_analogies_finnish.py

questions_file_path = "/tmp/questions-words_finnish.txt"

pairs = [
    {
        "name": "capital-common-countries",
        "items": [
            ("Helsinki", "Suomi"),
            ("Tukholma", "Ruotsi"),
            ("Moskova", "Venäjä"),
            ("Oslo", "Norja"),
            ("Kööpenhamina", "Tanska"),
            ("Reykjavik", "Islanti"),
            ("Tallinna", "Viro"),
            ("Riika", "Latvia"),
            ("Vilna", "Liettua"),
            ("Varsova", "Puola"),
            ("Berliini", "Saksa"),
            ("Lontoo", "Englanti"),
            ("Madrid", "Espanja"),
            ("Pariisi", "Ranska"),
            ("Rooma", "Italia"),
            ("Kiova", "Ukraina"),
            ("Ankara", "Turkki"),
            ("Bukarest", "Romania"),
```

```
        ("Amsterdam", "Hollanti"),
        ("Bryssel", "Belgia"),
        ("Ateena", "Kreikka"),
        ("Bern", "Sveitsi"),
        ("Budapest", "Unkari"),
    ]
},
{
    "name": "capital-world",
    "items": [
        ("Abuja", "Nigeria"),
        ("Accra", "Ghana"),
        ("Alger", "Algeria"),
        ("Amman", "Jordania"),
        ("Ankara", "Turkki"),
        ("Antananarivo", "Madagaskar"),
        ("Apia", "Samoa"),
        ("Asgabat", "Turkmenistan"),
        ("Asmara", "Eritrea"),
        ("Nur-Sultan", "Kazakstan"),
        ("Ateena", "Kreikka"),
        ("Bagdad", "Irak"),
        ("Baku", "Azerbaidzan"),
        ("Bamako", "Mali"),
        ("Bangkok", "Thaimaa"),
        ("Banjul", "Gambia"),
        ("Peking", "Kiina"),
        ("Beirut", "Libanon"),
        ("Belgrad", "Serbia"),
        ("Belmopan", "Belize"),
        ("Berliini", "Saksa"),
        ("Bern", "Sveitsi"),
        ("Biskek", "Kirgisia"),
        ("Bratislava", "Slovakia"),
        ("Bryssel", "Belgia"),
        ("Bukarest", "Romania"),
        ("Budapest", "Unkari"),
        ("Gitega", "Burundi"),
        ("Kairo", "Egypti"),
        ("Canberra", "Australia"),
        ("Caracas", "Venezuela"),
        ("Chisinau", "Moldova"),
        ("Conakry", "Guinea"),
        ("Kööpenhamina", "Tanska"),
```

```
("Dakar", "Senegal"),
("Damaskos", "Syyria"),
("Dhaka", "Bangladesh"),
("Doha", "Qatar"),
("Dublin", "Irlanti"),
("Dusanbe", "Tadzikistan"),
("Funafuti", "Tuvalu"),
("Gaborone", "Botswana"),
("Georgetown", "Guyana"),
("Hanoi", "Vietnam"),
("Harare", "Zimbabwe"),
("Havanna", "Kuuba"),
("Helsinki", "Suomi"),
("Islamabad", "Pakistan"),
("Jakarta", "Indonesia"),
("Kabul", "Afganistan"),
("Kampala", "Uganda"),
("Kathmandu", "Nepal"),
("Khartum", "Sudan"),
("Kiova", "Ukraina"),
("Kigali", "Ruanda"),
("Kingston", "Jamaika"),
("Libreville", "Gabon"),
("Lilongwe", "Malawi"),
("Lima", "Peru"),
("Lissabon", "Portugali"),
("Ljubljana", "Slovenia"),
("Lontoo", "Englanti"),
("Luanda", "Angola"),
("Lusaka", "Sambia"),
("Madrid", "Espanja"),
("Managua", "Nicaragua"),
("Manama", "Bahrain"),
("Manila", "Filippiinit"),
("Maputo", "Mosambik"),
("Minsk", "Valko-Venäjä"),
("Mogadishu", "Somalia"),
("Monrovia", "Liberia"),
("Montevideo", "Uruguay"),
("Moskova", "Venäjä"),
("Masqat", "Oman"),
("Nairobi", "Kenia"),
("Nassau", "Bahama"),
("Niamey", "Niger"),
```

```
        ("Nikosia", "Kypros"),
        ("Nouakchott", "Mauritania"),
        ("Nuuk", "Grönlanti"),
        ("Oslo", "Norja"),
        ("Ottawa", "Kanada"),
        ("Paramaribo", "Suriname"),
        ("Pariisi", "Ranska"),
        ("Podgorica", "Montenegro"),
        ("Quito", "Ecuador"),
        ("Rabat", "Marokko"),
        ("Riika", "Latvia"),
        ("Rooma", "Italia"),
        ("Roseau", "Dominica"),
        ("Santiago", "Chile"),
        ("Skopje", "Makedonia"),
        ("Sofia", "Bulgaria"),
        ("Tukholma", "Ruotsi"),
        ("Suva", "Fidzi"),
        ("Taipei", "Taiwan"),
        ("Tallinna", "Viro"),
        ("Taskent", "Uzbekistan"),
        ("Tbilisi", "Georgia"),
        ("Tegucigalpa", "Honduras"),
        ("Teheran", "Iran"),
        ("Thimphu", "Bhutan"),
        ("Tirana", "Albania"),
        ("Tokio", "Japani"),
        ("Tripoli", "Libya"),
        ("Tunis", "Tunisia"),
        ("Vaduz", "Liechtenstein"),
        ("Valletta", "Malta"),
        ("Wien", "Itävalta"),
        ("Vientiane", "Laos"),
        ("Vilna", "Liettua"),
        ("Varsova", "Puola"),
        ("Windhoek", "Namibia"),
        ("Jerevan", "Armenia"),
        ("Zagreb", "Kroatia")
    ]
},
{
    "name": "currency",
    "items": [
        ("Algeria", "dinaari"),
```

```
        ("Angola", "kwanza"),
        ("Argentiina", "Peso"),
        ("Armenia", "dram"),
        ("Brasilia", "real"),
        ("Bulgaria", "leva"),
        ("Kambodza", "riel"),
        ("Kanada", "dollari"),
        ("Kroatia", "kuna"),
        ("Tanska", "kruunu"),
        ("EU", "euro"),
        ("Unkari", "forintti"),
        ("Intia", "rupia"),
        ("Iran", "rial"),
        ("Japani", "jeni"),
        ("Etelä-Korea", "won"),
        ("Makedonia", "denaari"),
        ("Malesia", "ringgit"),
        ("Meksiko", "peso"),
        ("Nigeria", "naira"),
        ("Puola", "zloty"),
        ("Romania", "leu"),
        ("Venäjä", "rupla"),
        ("Ruotsi", "kruunu"),
        ("Thaimaa", "baht"),
        ("Ukraina", "hryvnia"),
        ("Yhdysvallat", "dollari"),
        ("Vietnam", "dong"),
    ]
},
{
    "name": "family",
    "items": [
        ("poika", "tyttö"),
        ("veli", "sisko"),
        ("veli", "sisar"),
        ("isä", "äiti"),
        ("isoisä", "isoäiti"),
        ("sulhanen", "morsian"),
        ("mies", "vaimo"),
        ("kuningas", "kuningatar"),
        ("mies", "nainen"),
        ("prinssi", "prinsessa"),
        ("poika", "tytär"),
        ("setä", "täti"),
```

```
    ]
},
{
    "name": "gram2-opposite",
    "items": [
        ("varma", "epävarma"),
        ("selvä", "epäselvä"),
        ("mukava", "epämukava"),
        ("tehokas", "tehoton"),
        ("eettinen", "epäeettinen"),
        ("onnekas", "epäonnekas"),
        ("rehellinen", "epärehellinen"),
        ("tunnettu", "tuntematon"),
        ("looginen", "epälooginen"),
        ("miellyttävä", "epämiellyttävä"),
        ("mahdollinen", "mahdoton"),
        ("rationaalinen", "irrationaalinen"),
        ("vastuullinen", "vastuuton"),
        ("varma", "epävarma"),
    ]
},
{
    "name": "gram6-nationality-adjective",
    "items": [
        ("Suomi", "suomalainen"),
        ("Ruotsi", "ruotsalainen"),
        ("Venäjä", "venäläinen"),
        ("Norja", "norjalainen"),
        ("Tanska", "tanskalainen"),
        ("Islanti", "islantilainen"),
        ("Viro", "virolainen"),
        ("Latvia", "latvialainen"),
        ("Liettua", "liettualainen"),
        ("Puola", "puolalainen"),
        ("Saksa", "saksalainen"),
        ("Englanti", "englantilainen"),
        ("Espanja", "espanjalainen"),
        ("Ranska", "ranskalainen"),
        ("Italia", "italialainen"),
        ("Ukraina", "ukrainalainen"),
        ("Turkki", "turkkilainen"),
        ("Romania", "romanialainen"),
        ("Hollanti", "hollantilainen"),
        ("Belgia", "belgialainen"),
```

```
            ("Kreikka", "kreikkalainen"),
            ("Sveitsi", "sveitsiläinen"),
            ("Unkari", "unkarilainen"),
            ("Kiina", "kiinalainen"),
            ("Intia", "intialainen"),
            ("Yhdysvallat", "amerikkalainen"),
            ("Indonesia", "indonesialainen"),
            ("Pakistan", "pakistanilainen"),
            ("Brasilia", "brasilialainen"),
            ("Nigeria", "nigerialainen"),
            ("Bangladesh", "bangladeshilainen"),
            ("Meksiko", "meksikolainen"),
            ("Japani", "japanilainen")
        ]
    }
]


def gen():
    for section in pairs:
        items = sorted(section["items"])
        if len(items) > 1:
            yield(f": {section['name']}")
            for i, (a, b) in enumerate(items):
                items_40 = (items[:i] + items[i+1:])[:40]
                for c, d in items_40:
                    yield(f"{a} {b} {c} {d}")


if __name__ == "__main__":
    with open(questions_file_path, "w", encoding="utf-8") as f:
        for line in gen():
            f.write(line+"\n")
```